HOCHSCHULE MITTWEIDA
University of Applied Sciences

MASTERARBEIT

Herr
Elias Ullrich

Migration serviceorientierter Anwendungen für den vertrauenswürdigen Betrieb in der Cloud mittels Intel SGX

2020

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Migration serviceorientierter Anwendungen für den vertrauenswürdigen Betrieb in der Cloud mittels Intel SGX

Autor:
Elias Ullrich

Studiengang:
Cybercrime / Cybersecurity

Seminargruppe:
CY18wC-M

Erstprüfer:
Prof. Dr.-Ing. Thomas Beierlein

Zweitprüfer:
Dr.-Ing. Stephan Groß

Dresden, November 2020

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Migrating Service Oriented Applications for Confidential Cloud Computing using Intel SGX Enclaves

Author:
Elias Ullrich

Course of Studies:
Cybercrime / Cybersecurity

Seminar Group:
CY18wC-M

First Referee:
Prof. Dr.-Ing. Thomas Beierlein

Second Referee:
Dr.-Ing. Stephan Groß

Dresden, November 2020

Bibliographische Angaben

Ullrich, Elias: Migration serviceorientierter Anwendungen für den vertrauenswürdigen Betrieb in der Cloud mittels Intel SGX. 170 Seiten, 14 Abbildungen, 7 Tabellen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften.

Masterarbeit, 2020

Referat

In der letzten Dekade hat sich die Verwendung von Cloud-Computing für das verlässliche und kostengünstige Betreiben von IT-Ressourcen und Applikationen weitgehend etabliert. Für Anwendungsszenarien mit hochsensitiven Daten können jedoch inakzeptable Restrisiken in Bezug auf deren Vertraulichkeit und Integrität – insbesondere während der eigentlichen Datenverarbeitung – verbleiben.
Mit Hilfe von Intel SGX ist die Entwicklung dahingehend abgesicherter Anwendungen möglich, eine Nutzung dessen Funktionalität durch bereits bestehende Applikationen dagegen nicht ohne Weiteres. Um diese vor Angriffen auf die Vertraulichkeit oder Integrität mittels SGX zu schützen, wurden in der Forschung verschiedene Ansätze zur Migration entwickelt. Insbesondere serviceorientierte, aus mehreren Einzeldiensten bestehende, Anwendungen bedürfen dafür jedoch einer ausführlichen Planung und strukturierten Vorgehens.
Die vorliegende Masterarbeit hat das Ziel, ein grundlegendes Verständnis für SGX zu vermitteln und die Migration solcher Anwendungen zu vereinfachen. Dafür wird eine generische Methodik vorgestellt sowie anhand des Beispiels einer populären Webanwendung hinsichtlich ihrer Funktionalität, Sicherheit und Performanz evaluiert. Zusätzlich erfolgt die Diskussion der Migration von Container-Anwendungen sowie deren Besonderheiten und Möglichkeiten zur Orchestrierung.

Bibliographic Information

Ullrich, Elias: Migrating Service Oriented Applications for Confidential Cloud Computing using Intel SGX Enclaves. 170 pages, 14 figures, 7 tables, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences.

Master Thesis, 2020

Abstract

In the last decade, the usage of cloud computing for highly-available and cost-efficient resource and application deployment grew rapidly. Still, there exist unacceptable risks concerning integrity and confidentiality of sensitive information – especially data in use.

In software development, Intel SGX can be utilized to secure this data, whereas adopting existent programs for its use is difficult. For this purpose, various scientific studies have been published dealing with the migration of non-SGX applications. Those composed of multiple services require a careful preparation and structured migration approach. Furthermore, one may need a combination of different tools and frameworks to achieve this objective.

In this Master Thesis, we give a fundamental introduction to Intel SGX and aim at simplifying the migration of legacy service-oriented applications by presenting a generic methodology. Additionally, we evaluate the functionality, security and performance of a popular web application after being ported using this methodology. Finally, we discuss a possible combination of containerized applications and SGX together with their orchestration.

Danksagungen

An dieser Stelle möchte ich mich bedanken bei all den lieben Menschen, die mich während der Bearbeitungszeit dieser Masterarbeit unterstützt und diese damit erst möglich gemacht haben.
Als Erstes danke ich meiner Frau, die mir in dieser Zeit (und weit darüber hinaus) eine wertvolle Stütze, hilfreiche Ratgeberin und professionelle Korrekturleserin war. Neben der häufigen Unterstützung beim Finden der richtigen Wörter, bin ich für jede Ermutigung sowie die richtige Portion Ablenkung und Motivation dankbar – insbesondere das äußerst motivierende Backwerk zur richtigen Zeit.
Weiterhin bedanke ich mich bei Professor Beierlein, der selbst unter besonderen Umständen viel Zeit in die Betreuung meiner Arbeit investierte und mir mit guter Erreichbarkeit und hilfreichen Ratschlägen unterstützend zur Seite stand.
Bei Stephan Groß möchte ich mich bedanken für alles, was ich durch ihn lernen konnte – über eine wissenschaftliche Arbeitsweise, die unterschiedlichsten technischen Zusammenhänge und das praktische Leben. Auch für die kurzfristige Hilfe in jeglicher Hinsicht, die realistische Einschätzung meiner (oft zu) großen Pläne sowie das so häufige „Zeit-frei-schaufeln“ im komplexen Arbeitsalltag bin ich sehr dankbar.
Zuletzt möchte ich meiner Familie Danke sagen, die mich nicht unwesentlich zu dem gemacht hat, der ich heute bin und der eine solche Arbeit abgeben kann. Insbesondere meinem Cousin danke ich für die sehr spontane und dennoch zügige Korrektur von fast 100 Seiten Text.

Allen voran möchte ich mich jedoch bei Jesus Christus bedanken, der mich geschaffen hat und mir ein Leben in Freiheit ermöglicht.
Er war es, der mir immer wieder neu Freude an der Arbeit und den Blick für das Wesentliche geschenkt, in den schwierigen Phasen eine neue Perspektive aufgezeigt und mir geholfen hat, die komplexen Zusammenhänge verstehen und richtig anwenden zu können. Ohne seine Hilfe hätte ich keine einzige Seite dieser Arbeit schreiben oder das Studium auch nur bis zu diesem Punkt absolvieren können. Ihm verdanke ich mein Leben sowie alle Fähigkeiten und Begabungen, die in dieser Arbeit zum Tragen kommen (nach [Die Bibel, Psalm 139, Vers 14]).

I Inhaltsverzeichnis

II Abbildungsverzeichnis

III Tabellenverzeichnis

IV Listingverzeichnis

V Abkürzungsverzeichnis

A_{Cont} ..... Anwendung in einem (Docker-) Container

eData ..... Daten einer SGX-Enklave

f_B ..... Bridge Function

$f_{B,t}$ ..... trusted Bridge Function

$f_{B,u}$ ..... untrusted Bridge Function

A_L ..... Nicht-SGX-Anwendung

$\mathcal{M}_u$ ..... untrusted Memory

A_{MS} ..... Multi Service Application

$A_{MS,Cont}$ ..... aus Containern bestehende Multi Service Application

$A_{L,p}$ ..... in das SGX-Umfeld portierte Nicht-SGX-Anwendung

$A_{MS,p}$ ..... in das SGX-Umfeld portierte Multi Service Application

$\mathcal{P}_t$ ..... trusted Part (einer SGX-Anwendung)

$\mathcal{P}_u$ ..... untrusted Part (einer SGX-Anwendung)

A_{SGX} ..... SGX-Anwendung

svc_i ..... einzelner Dienst als Teil einer Multi Service Application

tCode ..... trusted Code

`<I>` ..... Installationsverzeichnis einer Anwendung

`<WPDir>` ..... Installationsverzeichnis der WordPress-Anwendung

AE ..... Architectural Enclave

AESM ..... Architectural Enclave Service Manager

AEX ..... Asynchronous Enclave Exit

AJP ..... Apache JServ Protocol

API ..... Application Programming Interface

AS ..... Anwendungsserver

BIOS ..... Basic Input/Output System

BSt ..... Backing Storage

CA ..... Certificate Authority

CCC ..... Confidential Computing Consortium

CGI ..... Common Gateway Interface

cgroup ..... Control Group

CMS ..... Content-Management-System

CN ..... Common Name

CPU ..... Central Processing Unit

CSS ..... Cascading Style Sheets

DB ..... Datenbank

DCAP ..... Data Center Attestation Primitives

DCAP-VS ..... Verifikationsserver für Attestation mit Data Center Attestation Primitives

ECall ..... Enclave Call

EPC ..... Enclave Page Cache

EPCM ..... Enclave Page Cache Map

EPID ..... Enhanced Privacy ID

FPM ..... FastCGI Process Manager

FSPF ..... (SCONE) File System Protection File

GCC ..... GNU Compiler Collection

GiB ..... Gibibyte

glibc ..... GNU C Library

GNU ..... GNU is not Unix

GSC ..... Graphene Shielded Containers

HTML ..... Hypertext Markup Language

HTTP ..... Hypertext Transfer Protocol

HTTPS ..... Hypertext Transfer Protocol Secure

I/O ..... Input/Output

IAS ..... Intel Attestation Service

iKGF ..... Intel Key Generation Facility

IPC ..... Inter-Process Communication

ISR ..... Interrupt Service Routine

KiB ..... Kibibyte

kLOC ..... 1000 Lines of Code

LE ..... Launch Enclave

libOS ..... Library Operating System

LOC ..... Lines of Code

MEE ..... Memory Encryption Engine

MiB ..... Mebibyte

OCall ..... Outside Call

OE ..... Owning Enclave (einer EPC-Seite)

OS ..... Operating System

PaaS ..... Platform-as-a-Service

PDF ..... Portable Document Format

PHP ..... PHP Hypertext Preprocessor

PRM ..... Processor Reserved Memory

PSE ..... Platform Service Enclave

PSW ..... Plattformsoftware

QE ..... Quoting Enclave

RAM ..... Random Access Memory

REST ..... Representational State Transfer

SCGI ..... Simple Common Gateway Interface

SDK ..... Software Development Kit

SGX ..... Software Guard Extensions

SHA ..... Secure Hash Algorithm

SRC ..... Secure Remote Computation

Syscall ..... Systemcall

Syscall-Komp. .. Systemcall-Kompatibilität

TB ..... Terabyte

TCB ..... Trusted Computing Base

TCP ..... Transmission Control Protocol

TEE ..... Trusted Execution Environment

TLB ..... Translation Lookaside Buffer

TLS ..... Transport Layer Security

UEFI ..... Unified Extensible Firmware Interface

VPN ..... Virtual Private Network

WebApp ..... Webanwendung

WS ..... Webserver

WSGI ..... Web Server Gateway Interface

1 Einleitung

Mehr und mehr Unternehmen sehen großes Potential in den Möglichkeiten von Cloud-Computing und setzen die entsprechenden Technologien in einem wesentlichen Teil ihrer IT-Infrastruktur ein. So haben in einer 2019 durchgeführten Studie 94% der befragten Unternehmen angegeben, von Cloud-Computing Gebrauch zu machen bzw. diesen zu planen oder zu diskutieren [Hei+20].
Neben den vielen Vorteilen, die eine Auslagerung von Rechnerressourcen und Anwendungen zu einem externen Cloud-Anbieter mit sich bringt[1], sehen sich Nutzer Cloud-basierter Ressourcen auch mit verschiedenen Risiken konfrontiert, welche bereits vor deren Einsatz diskutiert und den Möglichkeiten gegenübergestellt werden sollten.

Für einige dieser Risiken können verschiedene Maßnahmen aus dem IT-Sicherheitsbereich zur Mitigation verwendet werden. So ist es beispielsweise möglich, für den Schutz der im Cloud-Computing verarbeiteten Daten vor externen Angreifern Firewalls bzw. Application Level Gateways [SH99, S. 6] einzusetzen, welche den Datenverkehr in virtuellen Netzwerken (sog. „Software-Defined Networks“ [Xia+14]) regulieren. Ergänzend ist auch die Verwendung komplexer Verfahren zur Nutzerüberprüfung wie Zwei-Faktor-Authentifizierung möglich.
Einige andere Risiken können jedoch nicht durch den Einsatz solcher „traditionellen“ Werkzeuge verringert werden. So wird z.B. in jedem Fall das Vertrauen des Nutzers in eine integre Handlungsweise des Betreibers einer Public-Cloud-Umgebung vorausgesetzt.
Ein Kunde muss grundsätzlich davon ausgehen, dass für den Anbieter die Möglichkeit besteht, vollen Zugriff auf alle in dieser Cloud-Umgebung präsenten Daten zu erlangen, da er die den virtuellen Ressourcen zugrundeliegende Hardware kontrolliert und für deren Verwaltung Administratoren beschäftigt. Deshalb ist das Vertrauen auf die Einhaltung der vertraglichen Bedingungen erforderlich, eine Kontrolle oder gar Vermeidung unerlaubter Zugriffe ist nicht möglich.
Eine ähnlich privilegierte Position wie solche „Innentäter“ besitzt ein externer Angreifer, dem es gelungen ist, Zugriff auf die Cloud-Systeme zu erlangen und – z.B.

[1] Vorteile des Cloud-Computings sind u.a. die Aufteilung der Verantwortlichkeiten nach dem „Shared Responsibility Model“, die nutzungsbasierte Kostenabrechnung und die sog. „Elastizität“, nach der die Provisionierung benötigter Ressourcen an veränderte Bedarfe umgehend angepasst wird (nach [MG11]).

durch eine Rechteausweitung – über die Berechtigungen eines Administrationskontos (unter Linux z.B. das des Benutzers *root*) zu verfügen. Auch in diesem Fall können IT-Sicherheitswerkzeuge wie die oben erwähnten wenig zur Mitigation beitragen, da sie in der Regel von einem Administrator verwaltet und von einem Angreifer mit dessen Rechten zur Begünstigung der Tat entsprechend deaktiviert werden können.
Besonders bei Anwendungsszenarien mit hochsensitiven Daten wie Geschäftsgeheimnissen oder personenbezogenen Daten kann dies ein inakzeptables Risiko darstellen, welches dem Einsatz von Cloud-Computing entgegensteht [Sul20].

1.1 Motivation

Um dem beschriebenen Risiko von Innentätern und privilegierten Angreifern zu begegnen, ist eine konsequente Sicherstellung der Vertraulichkeit und Integrität schützenswerter Daten während deren gesamten Lebenszyklus erforderlich. Demnach müssen solche Daten (1.) bei der Übertragung (engl.: Data in transit), (2.) während der Verarbeitung (engl.: Data in use) und (3.) in auf einem Datenträger gespeicherter Form (engl.: Data at rest) durch technische Maßnahmen geschützt werden.
Während zur Lösung der Fälle (1.) und (3.) eine Vielzahl von Werkzeugen und Verfahren existiert, stellt der Schutz von Daten bei der Verarbeitung eine besondere Herausforderung dar, vor allem wenn diese auf einem entfernten, nicht vollständig kontrollierbaren Rechner stattfindet.

Eine der Technologien zur Absicherung von *Data in use* sind die vom Prozessorhersteller Intel entwickelten und in der Rechnerhardware verankerten „Software Guard Extensions“ (kurz: SGX). Diese Erweiterung des CPU-Befehlssatzes ermöglicht die Durchführung von Rechenoperationen unter Ausschluss des Betriebssystems und anderer Programme, was auch privilegierten Benutzern keinen Zugriff auf die verarbeiteten Daten gewährt.
Mit Hilfe von Intel SGX ist es möglich, die von einer Anwendung preisgegebene Angriffsfläche zu reduzieren, indem die Erforderlichkeit des Vertrauens in das Betriebssystem (und dessen administrierende Nutzer) eliminiert wird und eine Absicherung auf Ebene der Hardware stattfindet.
Dabei zielt SGX explizit auf die Schutzziele Vertraulichkeit und Integrität ab, nicht aber auf die Verfügbarkeit. Letztere kann zwar auch durch Innentäter oder privilegierte Angreifer gefährdet werden, für ihre Sicherstellung existieren aber verschiedene weitere Maßnahmen, die bereits umfassend zum Einsatz kommen, weshalb keine

weiterführende Maßnahme durch SGX benötigt wird[2].

Mit speziell für den Einsatz im Intel-SGX-Szenario entwickelter Software kann die Ausführung von sicherheitskritischen Anwendungen am besten geschützt werden. In der Praxis ist allerdings auch für bereits existierende Applikationen eine Steigerung der Sicherheit ohne die Erforderlichkeit zeit- und kostenintensiver (Neu-) Entwicklungen erwünscht.
Zur Unterstützung einer solchen Migration von einfachen Anwendungen (wie z.B. eines in C++ geschriebenen Programms) in das von SGX abgesicherte Umfeld wurden verschiedene Ansätze entwickelt, die in Abschnitt 2.2 erläutert und gegenübergestellt werden. Viele Anwendungen (z.B. die immer mehr eingesetzten Webanwendungen) weisen allerdings einen komplexeren Aufbau auf und bestehen aus mehreren Einzeldiensten, die miteinander in Interaktion treten, um die gesamte Funktionalität bereitzustellen. Zur Absicherung solcher „serviceorientierter“ Anwendungen mit Hilfe von Intel SGX sind weitere Analysen sowie eine umfassende Planung und strukturierte Durchführung der Migration notwendig.
Weiterhin wird zur Bereitstellung einzelner Dienste und ganzer Applikationen – insbesondere in Cloud-Umgebungen – vermehrt das Containerformat verwendet. Soll die Funktionalität von SGX bei Anwendungen zum Einsatz kommen, welche in einer Containerumgebung ausgeführt werden, müssen deren Besonderheiten während der Migration Berücksichtigung finden.

1.2 Zielstellung

Die vorliegende Arbeit soll zur Erleichterung der Migration von bestehenden (insbesondere serviceorientierten) Anwendungen in das Intel-SGX-Umfeld beitragen.
Zu diesem Zweck werden zuerst die Software Guard Extensions vorgestellt und ihre grundlegende Funktionsweise erläutert. Außerdem erfolgt eine Analyse der existierenden wissenschaftlichen Studien über eine solche Anwendungsportierung sowie deren zugrundeliegenden Strategien und Schwerpunktsetzung. Jede dieser Herangehensweisen weist ihre eigenen Vor- und Nachteile auf, die gegeneinander abgewogen werden müssen, um die für das jeweilige Anwendungsszenario am besten geeignete zu bestimmen.

[2] Zur Absicherung der Verfügbarkeit eines im Internet erreichbaren Webdienstes kann z.B. eine redundante Netzwerkarchitektur in Kombination mit einem vorgeschalteten Lastenausgleich (engl.: Load Balancer) verwendet werden.
Der Anbieter einer Cloud-Umgebung hat zwar auch auf diesen Zugriff und könnte die Verfügbarkeit nachteilig beeinflussen, aufgrund fehlenden finanziellen Nutzens dieser Vertragsverletzung ist dieses Risiko in der Regel jedoch als gering einzustufen.

Um serviceorientierte Applikationen auf die Ausführung im SGX-Umfeld vorzubereiten, wird eine generische Methodik aufgezeigt, welche deren strukturierte Analyse und die Entwicklung eines detaillierten Migrationsplans vereinfachen soll. Anhand eines Praxisbeispiels wird die Funktionsweise der entwickelten Methodik schließlich evaluiert und veranschaulicht sowie der Einfluss von Intel SGX auf die Funktionalität und Ausführungsgeschwindigkeit der Applikation herausgestellt.
Aufgrund der starken Präsenz von Containervirtualisierung im Umfeld von Cloud-Computing soll außerdem untersucht werden, wie sich deren Vorhandensein auf die Migration von enthaltenen Anwendungen auswirkt. Dabei werden die Besonderheiten von Containern diskutiert und Unterschiede zur vorgestellten Migrationsmethodik aufgezeigt.

1.3 Aufbau der Arbeit

Um die Möglichkeiten und Besonderheiten sowie Erforderlichkeiten der Intel Software Guard Extensions verstehen zu können, werden in Abschnitt 2.1 zunächst deren grundlegende Funktionsweise sowie die wichtigsten sicherheitsrelevanten Funktionen beschrieben. Danach erfolgt in Abschnitt 2.2 eine Einordnung und Erläuterung vorhandener Studien sowie kommerzieller Angebote bezüglich des Schutzes von bestehenden Applikationen durch SGX.
Darauf aufbauend wird in Kapitel 3 die methodische Vorgehensweise zur Migration serviceorientierter Anwendungen vorgestellt und anhand von Beispielen erläutert.
Zur Veranschaulichung und Evaluation dieser Methodik wird in Abschnitt 4.1 die Migration einer beispielhaften Webanwendung, des Open Source Content-Management-Systems „WordPress“, in ihren Einzelheiten beschrieben. Um die Einflüsse der sicheren Ausführungsumgebung auf deren Performanz zu bestimmen, wurden verschiedene Tests durchgeführt, deren Details und Ergebnisse in Abschnitt 4.2 aufgezeigt werden.
In Kapitel 5 wird abschließend die Migration von in Containern ausgeführten Anwendungen beleuchtet. Nach einer Vorstellung der grundlegenden Funktionsweise von Containervirtualisierung erfolgt hierbei eine Diskussion der Besonderheiten und Möglichkeiten bei der Migration solcher Anwendungen sowie der Verwendung von Orchestrierungswerkzeugen.

2 Grundlagen

2.1 Intel SGX

Sicheres Rechnen auf entfernten Rechnern (engl.: Secure Remote Computation, kurz: SRC) beschreibt die Möglichkeit, vertrauenswürdigen (i.d.R. eigenen) Code zur Verarbeitung vertraulicher Daten in einer nicht vertrauenswürdigen Umgebung (Hardware, Betriebssystem sowie Administratoren) unter Wahrung der Schutzziele Vertraulichkeit und Integrität auszuführen. In der Regel befindet sich dabei die Rechenumgebung unter Kontrolle einer fremden Partei, der nicht (vollumfänglich) vertraut wird. SRC stellt schon seit einer langen Zeit eine Herausforderung für die IT-Sicherheit dar [CLD17, S. 2 f.].
Homomorphe Verschlüsselung (z.B. das Verfahren von Gentry [Gen09]), ein existierender Lösungsansatz für Secure Remote Computation, erlaubt die Anwendung von Funktionen auf verschlüsselte Daten, wobei sowohl die Eingabe- als auch die Ausgabedaten nie entschlüsselt vorliegen. Die nachfolgende Entschlüsselung enthüllt aufgrund der mathematischen Eigenschaften des eingesetzten Verfahrens dennoch das richtige Ergebnis [Gen09, S. 5]. Dieses Verhalten kann durch die folgende Äquivalenz ausgedrückt werden.

$$\begin{gathered}\text{Klartexteingabe} \rightarrow \text{Verschlüss.} \rightarrow \text{Berechnung} \rightarrow \text{Entschlüss.} \rightarrow \text{Klartextausgabe} \\ \equiv \\ \text{Klartexteingabe} \rightarrow \text{Berechnung} \rightarrow \text{Klartextausgabe}\end{gathered}$$

Der Nachteil an homomorpher Verschlüsselung ist allerdings der signifikante Einfluss auf die Performanz, der ihrem praktischen Einsatz in den meisten Fällen entgegensteht [NLV11].

Einen weiteren Ansatz stellt der Einsatz von spezieller („trusted“) Hardware im entfernten Rechner dar, welche als „Trusted Execution Environment“ (kurz: TEE) [SAB15] bezeichnet wird und für die Sicherstellung von Datenvertraulichkeit und -integrität verantwortlich ist. Dadurch wird kein vollständiges Vertrauen in die Rechenumgebung, sondern nur in die Funktionsweise (und damit den Hersteller) der speziellen Hardwarekomponente benötigt.

Als einer von mehreren Anbietern setzt der Prozessorhersteller Intel den Ansatz einer TEE seit 2015[1] in seinen CPUs durch die „Software Guard Extensions“ um. Dabei handelt es sich um eine Erweiterung des Prozessorbefehlssatzes, welche bereits 2013 vorgestellt wurde [Hoe13]. SGX isoliert die Ausführung von vertrauenswürdigem Code demnach auf Hardwareebene, was ein Unterbinden von unerlaubten Zugriffen durch nicht vertrauenswürdige Stellen während der gesamten Berechnung ermöglicht [CLD17, S. 3 f.].

2.1.1 Bedrohungsmodell

Intel SGX wurde unter der Annahme eines bestimmten Bedrohungsmodells entwickelt, welches nachfolgend kurz vorgestellt wird.
In diesem Modell werden insbesondere mögliche Angriffe auf die Schutzziele Integrität und Vertraulichkeit schützenswerter Daten einer Anwendung – und u.U. auch deren Code – berücksichtigt. Das Schutzziel der Verfügbarkeit wird dagegen als nicht gefährdet angesehen. Weiterhin wird von der Möglichkeit ausgegangen, dass ein Angreifer, welcher auf solche Daten zugreifen möchte, die folgenden Bestandteile einer Ausführungsumgebung beeinflussen und für seine Zwecke manipulieren kann:

- die physische Hardware (mit Ausnahme der CPU) sowie alle Ein- und Ausgabeschnittstellen des Rechners
- das komplette Betriebssystem und damit alle Aktionen des Kernels (z.B. Entscheidungen über die Speicherverwaltung)
- privilegierte wie nicht-privilegierte Nutzerkonten
- jegliche weitere Anwendungen
- alle Bestandteile der Anwendung selbst, die sich außerhalb der geschützten SGX-Umgebung befinden

Demnach wird lediglich die SGX-Technologie innerhalb des Prozessors sowie die geschützte Anwendung selbst als sicher und vertrauenswürdig angesehen [CLD17; Pri+20].

[1] Intel SGX wurde zum ersten Mal in den 2015 veröffentlichten „Skylake“-Prozessoren eingesetzt [Win15].

2.1.2 Grundlegende Funktionsweise

Zur Realisierung einer Isolation zwischen sensitivem Code (bzw. Daten) und der nicht vertrauenswürdigen Umgebung schützt Intel SGX Ersteren in sicheren „Enklaven“ (engl.: Secure Enclaves), welche private, durch den Prozessor abgeschirmte und exklusiv nutzbare Speicherbereiche bereitstellen.
Durch die Durchsetzung der Zugriffsregeln auf Hardwareebene kann selbst ein privilegierter Benutzer, Treiber oder das Betriebssystem (engl.: Operating System, kurz: OS) nicht auf den Speicher einer Enklave zugreifen. Dieser wird „Enclave Page Cache“ (kurz: EPC) genannt und in Abschnitt 2.1.4.1 genauer erläutert [CD16, S. 2].

Eigenschaften und Aufbau Zusammengefasst bietet SGX die folgenden sicherheitsrelevanten Eigenschaften (nach [CD16, S. 2 f.] und [Geo19a]):

- Zugriff auf ihren Speicher ist nur für die jeweilige Enklave selbst möglich.
- Anwendungen im SGX-Modus *Production*[2] können nicht durch Debugger untersucht werden.
- Zugriffe auf Enklaven (z.B. Funktionsaufrufe) sind nur unter kontrollierten Randbedingungen möglich.
- Um eine Datenpreisgabe zu verhindern, können Enklaven keine Systemcalls aufrufen oder Interrupts behandeln.
- Alle Daten einer Secure Enclave werden beim Verlassen jener durch die CPU verschlüsselt.
- Kryptografische Schlüssel werden von nicht lesbaren, im Prozessor enthaltenen *Root Keys* abgeleitet.
- Die Integrität des in Enklaven ausgeführten Codes kann durch „Attestation“-Verfahren sichergestellt werden (siehe auch Abschnitt 2.1.6.1).

Damit eine schützenswerte Daten verarbeitende SGX-Anwendung A_{SGX} Vertraulichkeit und Integrität mittels Secure Enclaves sicherstellen kann, muss sie sich aus zwei Teilen zusammensetzen. Die erste, vertrauenswürdige, Komponente (im Folgenden als *trusted Part*, kurz: $\mathcal{P}_t$, bezeichnet) kann auf Geheimnisse zugreifen und wird in der Enklave ausgeführt. Sie beinhaltet deren Anweisungen (*trusted Code*, kurz: *tCode*) und jegliche enklaveninterne Daten (*Enclave Data*, kurz: *eData*). Der zweite,

[2] Eine Enklave kann in einem der vier SGX-Modi *Production*, *Pre-release*, *Debug* oder *Simulation* ausgeführt werden (wobei Letzterer keine realen SGX-Enklaven nutzt). Weitere Informationen können [JBZ16] entnommen werden.

nicht vertrauenswürdige, Teil (*untrusted Part*, kurz: $\mathcal{P}_u$), wird dagegen als normaler Systemprozess ausgeführt, ist für das Starten von $\mathcal{P}_t$ in einer oder mehreren Enklaven zuständig und enthält alle weitere, nicht schützenswerte Applikationslogik [Ada18a]. Dieser Aufbau wird in Abbildung 2.1 zusammengefasst.

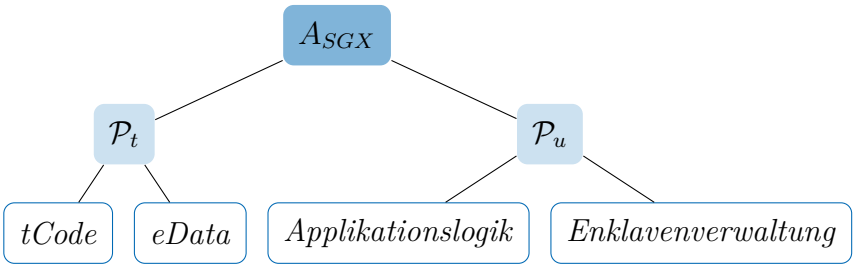


Abbildung 2.1: Aufbau einer SGX-Anwendung

Der Prozess einer Partitionierung in diese zwei Teile ist besonders bei bestehenden Anwendungen sehr komplex und aufwändig und wird in Abschnitt 2.1.3 weiter thematisiert.
Da der $\mathcal{P}_t$ also Teil einer Applikation ist, wird er im gleichen Prozess ausgeführt wie die A_{SGX} selbst und kann somit auch auf den Speicher vom $\mathcal{P}_u$ zugreifen, jedoch nicht umgekehrt [Ada18a].
Außerdem gilt zu beachten, dass SGX die Anzahl der Enklaven einer Anwendung nicht limitiert. Der $\mathcal{P}_t$ kann also auch aus mehreren gleichberechtigten Enklaven bestehen, wobei dies in der Praxis allerdings selten der Fall ist [Int20k, S. 8].

Ablauf einer SGX-Anwendung Der schematische Ablauf einer A_{SGX} wird in Abbildung 2.2 veranschaulicht. Der Einstiegspunkt einer solchen Anwendung befindet sich dabei stets im *untrusted Part* (in der Regel eine `main()`- oder vergleichbare Funktion), da eine Enklave durch die Anwendung selbst mit CPU-Unterstützung vorbereitet werden muss [MO16].

Nach eventuellen weiteren Programmschritten erfolgt im $\mathcal{P}_u$ die Initialisierung der Enklave, wobei der Prozessor deren Code, Daten sowie alle für SGX benötigte Informationen vorbereitet. Nach [CD16, S. 63 f.] kann dieser Vorgang durch die folgenden drei Schritte zusammengefasst werden.

1. Reservierung und Vorbereitung des benötigten Speichers im Enclave Page Cache
2. Laden von *tCode* und *eData* in den EPC sowie Aktualisierung der Checksummen der Enklave zum Umfassen aller Speicherinhalte
3. Abschluss der Initialisierung sowie Verifikation der Enklaven- und Autorensignatur

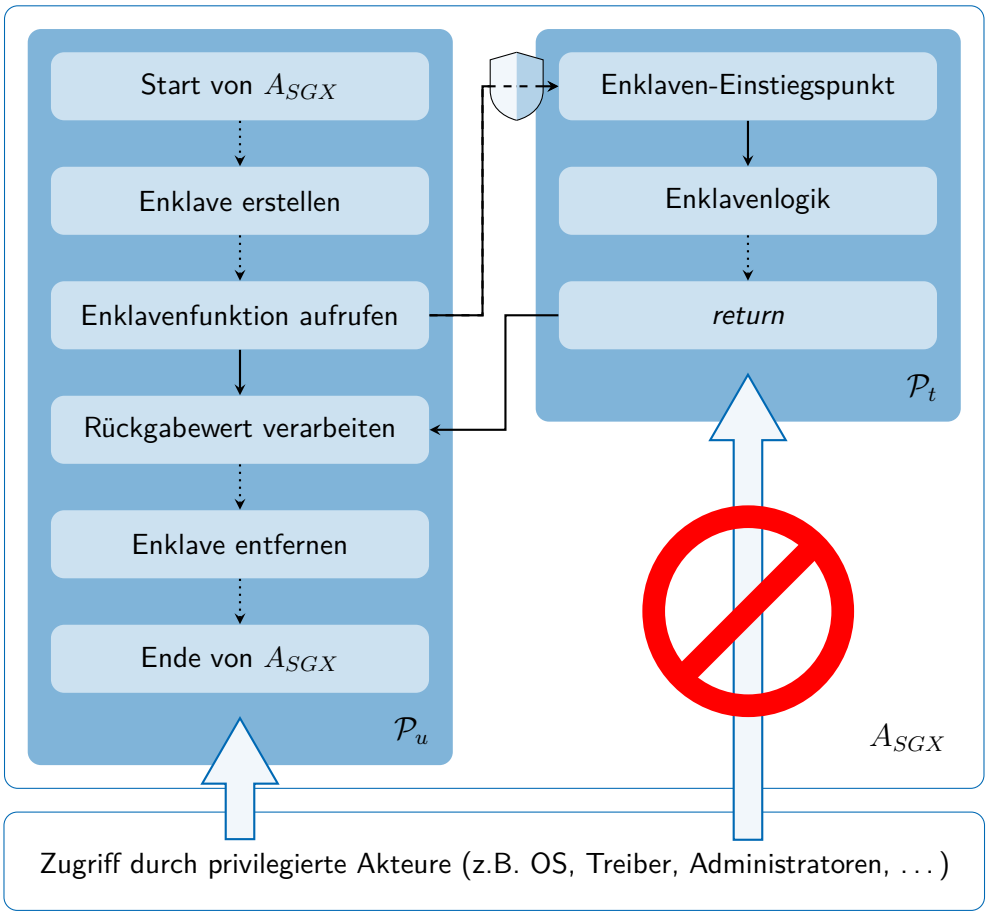


Abbildung 2.2: Schematischer Ablauf einer SGX-Anwendung

Wurde die Enklave erfolgreich angelegt, kann an beliebiger Stelle des Programms eine vorher definierte Enklavenfunktion aufgerufen und die Kontrolle nach umfassenden Sicherheitsüberprüfungen durch die CPU an den *tCode* übergeben werden. Außerdem wird der entsprechende logische Prozessor in den speziellen Betriebsmodus „Enclave Mode“ versetzt, der die Voraussetzung für jeglichen EPC-Zugriff ist.
Nachdem der vorgegebene Code im $\mathcal{P}_t$ ausgeführt wurde, verlässt der Prozessor den Enclave Mode sowie die Enklave wieder und der *untrusted Part* der Anwendung kann seine Ausführung unter Nutzung der eventuellen Rückgabewerte aus der Enklave fortsetzen [CD16, S. 65 ff.].
Im Falle von Hardware-Interrupts oder Speicherzugriffsfehlern während des Enclave Modes führt der Prozessor einen asynchronen Ausstieg aus der Enklave (engl.: Asynchronous Enclave Exit, kurz: AEX) durch[3] und überlässt die Behandlung der Interrupt Service Routine (kurz: ISR) des Betriebssystems und schließlich dem $\mathcal{P}_u$ der Applikation, welcher die Ausführung des unterbrochenen *tCodes* durch ein erneutes

[3]Bei einem AEX sichert die CPU den aktuellen Ausführungskontext (u.a. Zustand der Register) sowie die Eigenschaften der Enklave in verschlüsselter Form und löscht alle ihr zugehörigen Informationen, um die Preisgabe sensitiver Daten zu verhindern [CD16, S. 67 f.].

Betreten der Enklave wieder fortsetzen kann [CD16, S. 67 f.].
Wird diese nicht mehr benötigt, veranlasst die Anwendung durch den *untrusted Part* ihre Beendigung und Entfernung. In diesem Fall löscht der Prozessor alle Enklavendaten und deallociert die zugehörigen EPC-Bereiche, um zuletzt ihre Metadaten zu entfernen und damit die Enklave zu terminieren. Danach können bei Bedarf weitere Anweisungen im $\mathcal{P}_u$ ausgeführt werden [CD16, S. 66 f.].

2.1.3 Entwurf einer SGX-Anwendung

Wird eine Anwendung gezielt für den Einsatz im Zusammenhang mit Intel SGX-Enklaven entwickelt, ist es möglich, die Besonderheiten dieser Umgebung von Beginn an zu berücksichtigen. Im Gegensatz dazu können bereits bestehende Applikationen nicht unverändert im SGX-Kontext verwendet werden. Hierzu sind in der Regel vorangehende Anpassungen notwendig, um die Verwaltung von sowie die Codeausführung in Enklaven zu ermöglichen.
Mehr Informationen zum letzteren Fall können Abschnitt 2.2 entnommen werden. An dieser Stelle soll dagegen das Vorgehen beim Entwurf neuer SGX-Anwendungen anhand der im offiziellen „Intel® Software Guard Extensions Tutorial“ [MR16] bereitgestellten Informationen beschrieben werden.

2.1.3.1 Anwendungspartitionierung

Bei der Entwicklung der A_{SGX} muss nach der formalen Anforderungsspezifikation die Partitionierung in die zwei Teile $\mathcal{P}_u$ und $\mathcal{P}_t$ erfolgen, um das Programm entsprechend dieser Aufteilung zu strukturieren und die Sicherheitsvorteile einer Enklave nutzen zu können. Nach [MR16] erfolgt diese Partitionierung in vier Schritten:

1. Identifikation von schützenswerten Daten
2. Ableitung der Quellen, Senken und Verarbeiter dieser Daten
3. Spezifikation des $\mathcal{P}_t$
4. Entwicklung der für Enklavenverwaltung und -kommunikation zuständigen Anwendungsbestandteile

Dieses Vorgehen folgt der Empfehlung des SGX-Entwicklerhandbuchs, und damit dem Grundsatz, den *tCode* so klein wie möglich zu halten, um durch die Verringerung der Angriffsfläche eine höhere Sicherheit zu erzielen [Int20k, S. 8]. Durch diese Reduktion

der „Trusted Computing Base“ (kurz: TCB)[4] wird außerdem ein geringerer Speicherbedarf im EPC erreicht. Dies ermöglicht zum einen bei gleichbleibender Gesamtgröße die gleichzeitige Existenz einer größeren Anzahl von Enklaven und erfordert zum anderen weniger zeitintensive Auslagerungsvorgänge von Enklaven- in Systemspeicher (mehr dazu in Abschnitt 2.1.4.1).
Dieses Ziel wird erreicht, indem der $\mathcal{P}_t$ um die schützenswerten Daten herum entworfen wird und die Funktionen des *tCodes* auf deren Verarbeitung beschränkt werden. Alle Berechnungen, die nicht von solchen Daten abhängen, werden in den $\mathcal{P}_u$ ausgelagert.

Die Begrenzung der Enklave kann anschließend so gewählt werden, dass ein für den konkreten Anwendungszweck angemessener Kompromiss zwischen den folgenden drei Zielen gefunden wird:

- möglichst vollständige Verarbeitung von schützenswerten Daten innerhalb der Enklave
- Minimieren der Anzahl solche Daten verwendender Komponenten und damit des *tCodes*
- Minimieren von Interaktionen mit und Abhängigkeiten vom $\mathcal{P}_u$-Code

Dabei muss berücksichtigt werden, dass die Preisgabe von sensitiven Daten aus der Enklave nicht immer verhindert werden kann. Da Secure Enclaves selbst keine Ein- oder Ausgabe-Operationen (kurz: I/O) durchführen können (siehe Abschnitt 2.1.7), müssen z.B. für den Nutzer am Bildschirm auszugebende Informationen dem Betriebssystem zur Verfügung gestellt werden [MR16].
In diesem Zusammenhang ist der Vorteil des Einsatzes von SGX in Public-Cloud-Umgebungen jedoch, dass jegliche Ein- und Ausgaben über das Netzwerk verschickt werden und z.B. mittels TLS (Transport Layer Security) noch in der Enklave verschlüsselt werden können.

2.1.3.2 Schnittstellen einer Enklave

Wie bereits in Abschnitt 2.1.2 erläutert, erfolgt jeglicher Zugriff auf eine SGX-Enklave durch den nicht vertrauenswürdigen Teil der ihr zugeordneten Anwendung. So wird dieser z.B. bei I/O-Zugriffen zum Einlesen der Eingabewerte und anschließenden Wei-

[4] Die TCB umfasst alle Hard- und Softwarekomponenten eines (SGX-) Systems, die als sicher betrachtet werden können und denen Vertrauen entgegengebracht wird, also u.a. den Prozessor und EPC, die SGX-Implementierung sowie den $\mathcal{P}_t$ einer SGX-Anwendung [Int20k, S. 10].
Da die Größe aller anderen Komponenten konstant bleibt, kann die der TCB als proportional zum Umfang des $\mathcal{P}_t$ angesehen werden.

terleiten an die Enklave benötigt. Nach der Verarbeitung nimmt der $\mathcal{P}_u$ die Ergebnisse schließlich wieder entgegen und gibt sie aus.
Für die Kommunikation zwischen den beiden Teilen einer A_{SGX} gibt es drei verschiedene Szenarien [Ada18b]:

ECall Soll eine Funktion in der Enklave durch Code im $\mathcal{P}_u$ aufgerufen werden, wird ein sog. „Enclave Call“ (kurz: ECall) durchgeführt. Dabei werden die Funktionsargumente durch den Prozessor an die Enklave weitergereicht. Die Anwendung wird pausiert und die CPU wechselt in den Enclave Mode.

OCall Als Gegenstück verwendet die Enklave nach der Fertigstellung ihrer Berechnungen einen „Outside Call“ (kurz: OCall), um den Enclave Mode wieder zu verlassen und die Kontrolle an die A_{SGX} zu übergeben, deren Ausführung direkt nach dem zugehörigen ECall fortgesetzt wird.

AEX Der Asynchronous Enclave Exit beendet die Ausführung von *tCode* nach einem Interrupt und wurde bereits in Abschnitt 2.1.2 thematisiert.

Der Entwurf von Schnittstellen zwischen der vertrauenswürdigen Enklave und dem misstrauten $\mathcal{P}_u$ muss stets unter der Annahme geschehen, dass jegliche Software außerhalb der TCB Informationen fälschen, unterdrücken und weitergeben kann. Demnach dürfen bei OCalls keine sensitiven Daten nach außen dringen und alle Parameter eines ECalls müssen durch *tCode* überprüft werden.
Während der Entwicklung einer SGX-Anwendung müssen deshalb alle für den Datenaustausch mit der Enklave genutzten Funktionen spezifiziert werden. Für die Übertragung der jeweiligen Informationen werden dabei sog. „Proxy Functions“ bzw. „Bridge Functions“ (kurz: f_B) genutzt [MR16]. Diese werden durch das SGX-Hilfsprogramm „Edger8r“ für jeden ECall und OCall automatisch generiert, je eine für die Seite des $\mathcal{P}_u$ ($f_{B,u}$) sowie des $\mathcal{P}_t$ ($f_{B,t}$) [Int20k, S. 14 ff.].

Soll aus der A_{SGX} heraus eine Enklavenfunktion ausgeführt werden, erfolgt dies nur indirekt; zuerst wird die generierte Funktion $f_{B,u}$ aufgerufen, die ihrerseits den ECall durchführt und $f_{B,t}$ innerhalb der Enklave startet. Die eigentliche Enklavenfunktion des *tCodes* wird schließlich von $f_{B,t}$ aufgerufen. Bei einem OCall wird dieses Vorgehen entsprechend invertiert [Geo19b].
Die Verwendung dieser zwei zusätzlichen Funktionen ist notwendig, damit beim Wechsel in den oder aus dem Enclave Mode die Umgebung korrekt vorbereitet werden kann. So müssen beispielsweise Funktionsargumente vom Arbeitsspeicher in den EPC bzw. zurück kopiert und auf vorgegebene Bedingungen – wie z.B. eine maximale Größe – geprüft werden.

Außerdem ist auf diese Weise der Zugriff auf zwei verschiedene Rückgabewerte möglich: den Erfolgsstatus der ECall/OCall-Operation als Ergebnis der f_B sowie den eigentlichen Rückgabewert der aufgerufenen Funktion, welcher über einen Pointer referenziert wird [Mec16].
Die Folge von Funktionsaufrufen bei ECalls und OCalls wird in Abbildung 2.3 veranschaulicht.

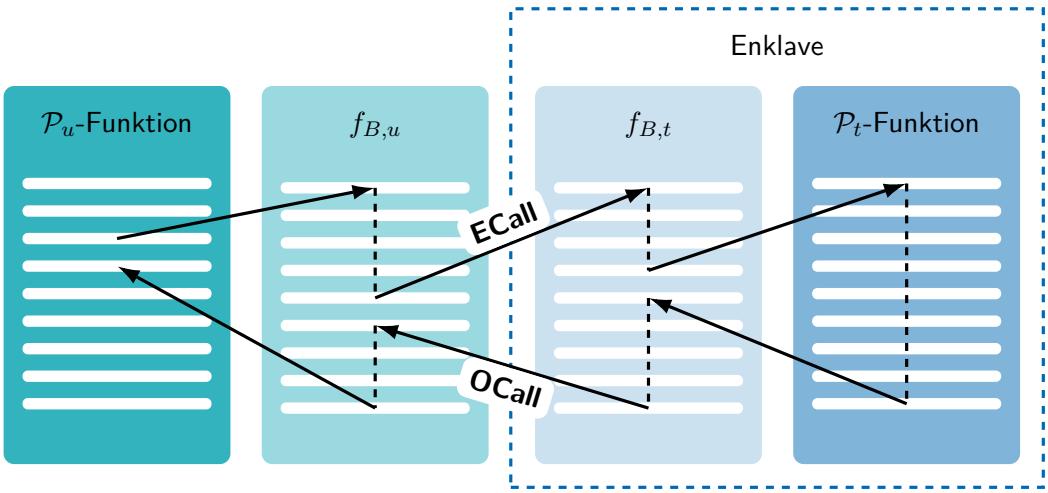


Abbildung 2.3: Funktionsablauf von ECalls und OCalls

2.1.4 SGX-Interna

Zum Verständnis der Funktionsweise und zur Einschätzung der Sicherheitsmerkmale von Intel SGX ist die Kenntnis und Einordnung einiger interner Komponenten notwendig. Dieser Abschnitt soll deshalb einen Überblick über die wichtigsten für SGX-Enklaven eingesetzten Konzepte und notwendigen Detailinformationen geben.
So wird in Abschnitt 2.1.4.1 der Enklaven-interne Hauptspeicher, der EPC, thematisiert, während Abschnitt 2.1.4.2 die wichtigsten zur Integritäts-, Authentizitäts- und Vertraulichkeitssicherung verwendeten kryptografischen Schlüssel und Daten vorstellen soll.

2.1.4.1 Enclave Page Cache

Der Enclave Page Cache (EPC) ermöglicht die prozessorgestützte Isolation von schützenswerten Daten und ist ein zentrales Element in der Umsetzung von SGX. In diesem Abschnitt sollen die wichtigsten Eigenschaften des EPC beschrieben werden. Weiterführende Informationen können z.B. dem offiziellen Software Developer's Manual

von Intel [Int20h] oder dem Artikel von Costan und Devadas [CD16] entnommen werden.

Einordnung Die Daten einer Secure Enclave können sich in drei verschiedenen Arten von Speichern befinden (nach [Vau+18, S. 731]):

Innerhalb der CPU werden Daten immer im Klartext abgelegt. Während sich der Prozessor im Enclave Mode befindet und die entsprechende Enklave ausführt, sind *eData* in den CPU-Registern sowie den Level-1- bis Level-3-Caches enthalten.

Der EPC bezeichnet den speziellen Bereich im Hauptspeicher, der als primärer Ablageort für die Programm- und Verarbeitungsdaten von SGX-Enklaven genutzt und vor dem Zugriff durch externe Ressourcen, wie dem Betriebssystem, geschützt wird. Der vorhandene Speicher wird dabei zwischen allen auf einem physischen Computer ausgeführten Enklaven aufgeteilt, diese können jedoch stets nur auf ihren eigenen zugreifen.

Andere RAM-Bereiche können ergänzend zum EPC ebenfalls für die Speicherung von *eData* genutzt werden, um Enklaven mit höherem Speicherbedarf zu unterstützen. Dazu lagert die CPU mit Hilfe des SGX-Treibers (mehr Informationen über diesen in Abschnitt 2.1.5) selten benutzte Speicherseiten des EPC in den restlichen Hauptspeicher aus, ähnlich zum Paging-Mechanismus von RAM-Seiten auf Swap-Speicher. Um die Vertraulichkeit der betreffenden Daten zu schützen, realisiert der Prozessor eine transparente Verschlüsselung beim Auslagern bzw. Entschlüsselung beim Rückschreiben in den EPC.

Aufbau des EPC Der Enclave Page Cache ist eine Untermenge des „Processor Reserved Memory“ (kurz: PRM) eines Computers. Dieser wiederum bezeichnet einen dedizierten Bereich im Hauptspeicher, welcher von der Firmware vor dem eigentlichen Systemstart reserviert und durch die CPU-interne „Memory Encryption Engine“ (kurz: MEE) während der gesamten Betriebsdauer verschlüsselt gehalten wird. Die Verschlüsselung ist für den Prozessor selbst transparent, während Anwendungen – insbesondere auch privilegierte Systemsoftware – nicht auf den Inhalt dieses Speicherbereichs zugreifen können, da nur der Prozessor selbst den PRM – mit Hilfe der MEE – entschlüsseln und auslesen kann [Min18, S. 8].
Die Größe des PRM kann im BIOS bzw. UEFI konfiguriert werden, ist aber durch

Hardwarelimitationen auf maximal 128 MiB[5] beschränkt, von denen bis zu 93,5 MiB für *eData* nutzbar sind [Vau+18, S. 731].

Der EPC selbst wird in Speicherseiten von je 4 KiB Größe unterteilt, die dynamisch belegt werden, jeweils aber maximal einer Enklave gleichzeitig zugeordnet sind, welche dann als „Owning Enclave“ (kurz: OE) bezeichnet wird.
Zur Verwaltung der EPC-Seiten und Überprüfung von Allokationen im Enklavenspeicher verwendet SGX eine weitere Struktur im PRM, die „Enclave Page Cache Map“ (kurz: EPCM), welche je einen Eintrag pro EPC-Speicherseite enthält. Dieser speichert beispielsweise den Status der Allokation sowie den Seitentyp und referenziert ihre OE [CD16, S. 58 f. & 92].
Der Aufbau des EPC und der Zusammenhang mit EPCM, PRM und dem System-RAM wird in Abbildung 2.4 zusammenfassend dargestellt.

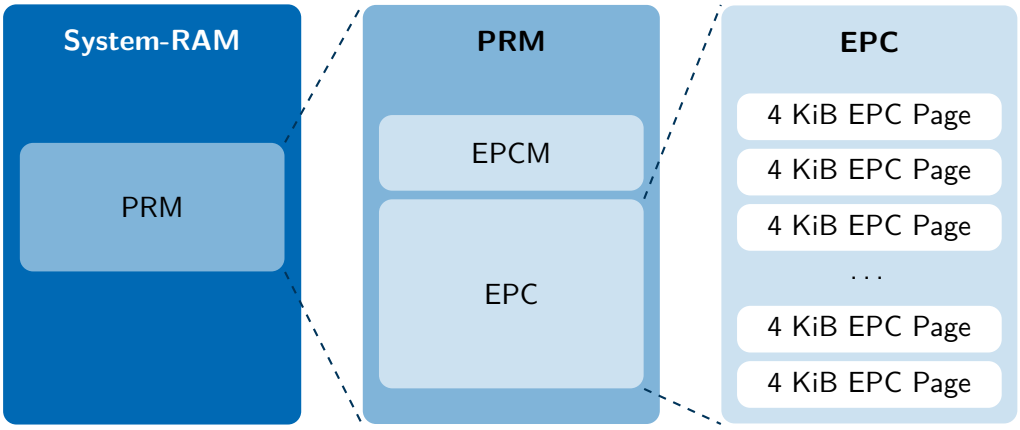


Abbildung 2.4: Aufbau und Lokation des EPC

Sicherheitsmechanismen Das Hauptziel der Verwendung eines dedizierten Enklavenspeichers wie des EPC liegt im Erreichen der Schutzziele Vertraulichkeit und Integrität. Demnach soll es nur einer Enklave selbst möglich sein, auf ihre (im RAM gespeicherten) Daten zuzugreifen. Mit Hilfe von SGX werden *eData* nur verschlüsselt im Hauptspeicher abgelegt und müssen durch die CPU-interne MEE vor der Nutzung wieder entschlüsselt werden. Weil dadurch eine Kontrolle aller Zugriffsversuche durch den SGX-Prozessor möglich ist, kann dieser ein exklusives Nutzungsrecht durch die jeweilige Enklave sicherstellen.
Die Speicherverwaltung (Allokation und Deallokation sowie Paging von Seiten) im EPC liegt – analog zum Rest des Hauptspeichers – in der Verantwortung des misstrauten Betriebssystems (bzw. Hypervisors). Letztlich werden aber alle Aktionen durch den Prozessor durchgeführt und erfordern die Verwendung spezieller SGX-Instruk-

[5]Mit den ab Ende 2019 [Int20m] eingeführten „Core“-Prozessoren der zehnten Generation verdoppelte Intel die Größe des PRM auf maximal 256 MiB [Int20a, S. 29].

tionen. Dieser kann z.B. mit Hilfe der EPCM sicherstellen, dass das Betriebssystem eine bereits allozierte EPC-Seite nicht einer weiteren Enklave zuweisen kann. Zuvor müsste die betreffende Seite erst von ihrer OE dealloziert werden, was ein Löschen und Überschreiben ihres Speicherinhalts durch die CPU zur Folge hätte. Da in der EPCM auch der Typ jeder Seite gespeichert wird, kann außerdem deren Missbrauch zu einem anderen als dem beabsichtigten Zweck verhindert werden.
Eine weitere Aufgabe des Betriebssystems ist das Laden des *tCodes* in die allozierten EPC-Seiten über ein CPU-Interface sowie die Bereitstellung von Diensten und Schnittstellen zum Management von Enklaven und deren Speicher für SGX-Anwendungen [CD16, S. 58 f.].
Soll eine EPC-Seite durch das OS in den restlichen RAM ausgelagert werden, schreibt SGX vor, dass alle logischen Prozessoren, die *tCode* aus der betreffenden OE bearbeiten, die Enklave verlassen müssen (in der Regel durch einen Interrupt, der zum AEX führt). Dadurch wird das Leeren des zugehörigen Translation Lookaside Buffers (kurz: TLB)[6] veranlasst, sodass keine Einträge mehr existieren, welche auf die nicht mehr vorhandene EPC-Seite verweisen würden. Vor dem Kopieren einer Seite in einen ungeschützten Speicherbereich, wird sie von der CPU verschlüsselt sowie mittels einer *Nonce*, der „page version“, vor Replay-Angriffen[7] geschützt [CD16, S. 69 ff.].

2.1.4.2 Kryptografie in SGX

Damit die Sicherheitsziele von Intel SGX erreicht werden können, kommen zahlreiche kryptografische Verfahren und Schlüssel bei unterschiedlichen Funktionen (z.B. „Attestation“ oder „Sealing“, siehe Abschnitt 2.1.6) zum Einsatz.
Da viele Implementierungsdetails eine hohe Komplexität aufweisen, sollen in diesem Abschnitt nur jene erläutert werden, die für einen Überblick der wichtigsten Schlüssel und Eigenschaften erforderlich sind. Weiterführende Informationen können z.B. dem Artikel von Costan und Devadas [CD16] oder dem offiziellen „Software Developer's Manual“ [Int20h] entnommen werden.

[6] Der TLB ist ein in der CPU (meist Teil der Memory Management Unit) integrierter Zwischenspeicher für Adresszuordnungen.
Beim Einsatz von virtuellem Speicher ist für RAM-Zugriffe stets eine Übersetzung von der virtuellen in die physische Adresse und dafür die Auswertung der Seitentabelle nötig, was zu einer verringerten Performanz führt. Der TLB wird genutzt, um häufig (bzw. zuletzt) erfragte Zuordnungen zwischenzuspeichern und durch den wegfallenden Umweg über die Seitentabelle den Zugriff zu beschleunigen [Tan09, S. 249].

[7] „Replay-Attacken“ stellen eine Klasse von Angriffen auf eine (kryptografische) Kommunikation dar. Ein entsprechender Angreifer fängt eine valide Nachricht zwischen zwei Kommunikationspartnern ab und unterdrückt oder versendet sie später – unverändert oder modifiziert – erneut (nach [Syv94, S. 1]).

Geheimnisse und Schlüssel Da die Sicherheit der Software Guard Extensions zum großen Teil auf kryptografischer Verschlüsselung basiert, werden im Lebenszyklus einer Enklave viele verschiedene, von der CPU generierte, Schlüssel verwendet (z.B. der `LaunchKey`, `ProvisioningKey` oder `AttestationKey`). An dieser Stelle sollen allerdings nur die Grundlagen vorgestellt werden, auf denen jegliche weitere Schlüssel basieren.
Während des Herstellungsprozesses einer SGX-fähigen CPU wird diese mit zwei Hardwareschlüsseln bestückt, die in unveränderbaren und nur vom Prozessor selbst lesbaren Speichermodulen abgelegt werden. Wie in [CD16, S. 85 f.] vorgeschlagen, werden sie im Folgenden als „Secrets“ bezeichnet, da sie in Verschlüsselungs- bzw. Signaturverfahren nie unverändert eingesetzt, sondern stets als „Wurzelschlüssel“ (engl.: Root Keys) zur Ableitung weiterer Schlüssel genutzt werden.

`ProvisioningSecret` Bei der Prozessorherstellung wird das `ProvisioningSecret` in einer dedizierten „Intel Key Generation Facility“ (kurz: iKGF) generiert und vom Hersteller in einer Datenbank gespeichert. Dieses, zwischen Intel und dem physischen Prozessor geteilte, Geheimnis wird z.B. zur Authentifizierung der Plattform gegenüber dem Hersteller benötigt. So kann ein Nutzer sichergehen, dass dieser nur korrekte SGX-Prozessoren als solche bescheinigt [Geo19c].

`SealSecret` Im Gegensatz dazu sichert Intel zu, dass alle Spuren des `SealSecrets` in der iKGF beseitigt werden und dieses ausschließlich dem Prozessor selbst bekannt ist. Alle Schlüssel, die nicht bei der Kommunikation mit dem Hersteller erforderlich sind, werden von diesem Root Key abgeleitet [CD16, S. 85 f.].

Da dem Hersteller demnach nur das `ProvisioningSecret` bekannt ist, kann er keine Rückschlüsse auf solche Schlüssel ziehen, welche (auch oder ausschließlich) vom `Seal Secret` abhängen (z.B. der `SealKey`) [CD16].

Daten und Datenstrukturen Neben diesen Schlüsseln werden bei SGX-Funktionen wie Attestation oder Sealing auch diverse, im EPC oder dem Prozessor selbst gespeicherte, (Meta-) Daten der Enklave bzw. der gesamten TCB eingesetzt. Dadurch kann eine kryptografische Bindung mit ihren Eigenschaften hergestellt werden. Außerdem kommen komplexe Datenstrukturen als Ein- oder Ausgabe verschiedener SGX-Prozesse zum Einsatz, welche jeweils mehrere dieser Informationen in sich vereinen.
Die für das weitere Verständnis notwendigen Daten und Datenstrukturen sollen an dieser Stelle kurz vorgestellt werden (nach [Ana+13, S. 2; CD16, S. 81 ff.]).

`MRENCLAVE` Identifiziert eine Enklave durch einen Hash über ihren *tCode*, bei ihrer Erstellung vorhandene *eData*, die Positionierung ihrer Speicherseiten im EPC sowie der für die Initialisierung verwendeten Flags.

`MRSIGNER` Identifiziert den Autor der Enklave durch einen Hash über seinen öffentlichen Schlüssel.

`REPORT` Dient bei der Local Attestation zum Authentizitätsnachweis einer Enklave gegenüber einer weiteren auf der selben Plattform.

`QUOTE` Wird bei der Remote Attestation zum Authentizitätsnachweis einer Enklave gegenüber einem externen Akteur erstellt und kann nur von einer speziellen Intel-Enklave signiert werden [Geo19c].

2.1.5 Die SGX-Umgebung

Sollen auf einem Rechner Anwendungen unter der Verwendung von Secure Enclaves ausgeführt werden, müssen neben dem Vorhandensein einer entsprechenden CPU auch mehrere Voraussetzungen auf Softwareseite erfüllt sein. Die Gesamtheit der für SGX verwendeten Softwarekomponenten wird hierbei als „SGX-Umgebung“ bezeichnet und in diesem Abschnitt erläutert.

BIOS-Einstellungen Um die SGX-Funktionen der CPU nutzen zu können, müssen in der Firmware die entsprechenden Einstellungen vorgenommen werden.
So kann SGX im BIOS eingeschaltet, ausgeschaltet oder eine softwaregesteuerte Aktivierung festgelegt werden. Im letzteren Fall ist der Zugriff auf SGX-Funktionen durch Anwendungen standardmäßig nicht möglich, er kann aber durch die Ausführung bestimmter Funktionen der Laufzeitbibliotheken im $\mathcal{P}_u$ freigeschaltet werden. Auf diese Weise muss der Prozessor den für SGX genutzten Speicherbereich des PRM – der vom Gesamtspeicher des Systems abgezogen wird – nicht grundsätzlich bereitstellen, sondern erst, wenn tatsächlich eine A_{SGX} ausgeführt wird [Mec17].
Zusätzlich ist im BIOS eine Anpassung der Größe des PRM (und damit auch des EPC) möglich, wobei der Maximalwert von 128 MiB jedoch nicht überschritten werden kann [Int17].

SGX-Treiber Für den Zugriff auf SGX-spezifische Hardwarefunktionen (nicht für den *Simulation*-Modus) wird ein eigener Treiber benötigt. Dieser wird von der Plattformsoftware instruiert, um mit der CPU zu interagieren und z.B. eine Enklave zu erstellen oder einen Wechsel in den Enclave Mode zu veranlassen. Unter Linux

wird er als Kernelmodul zum Nachladen, unter Windows dagegen als Treiberpaket bereitgestellt [Int20e; Int20k].

Plattformsoftware Die Intel SGX Plattformsoftware (kurz: PSW) umfasst mehrere Softwarekomponenten, stellt die Laufzeitumgebung für sichere Enklaven bereit und ist deshalb für jede Ausführung von SGX-Anwendungen erforderlich.
Neben Bibliotheken, die für das Starten und Verwalten von Enklaven benötigt werden und mit dem Treiber kommunizieren, enthält die PSW mehrere „Architectural Enclaves“ (kurz: AEs) sowie den Architectural Enclave Service Manager (kurz: AESM), einen Systemdienst zum Zugriff auf die benötigten AEs durch die A_{SGX}.
Intel stellt eine Vielzahl von signierten AEs bereit, um Verwaltungsaufgaben bei der Ausführung eigener SGX-Enklaven ebenfalls in einer durch Hardware geschützten Umgebung durchzuführen [Ada18b; Geo19d; Sca+19, S. 3]. Die für diese Arbeit relevanten Architectural Enclaves werden im Folgenden vorgestellt.

Launch Enclave (LE) Sie wird für den Start aller anderen Enklaven benötigt und erlaubt diesen erst nach der Überprüfung aller Signaturen und Checksummen.

Quoting Enclave (QE) Diese Enklave ist für die Remote Attestation verantwortlich, bei der sie einen lokal erstellten `REPORT` signiert und so bestätigt.

Platform Service Enclaves (PSEs) Sie fassen mehrere AEs zusammen, die anderen Enklaven verschiedene weitere Dienste bereitstellen, wie z.B. monotone Zähler oder kryptografische Funktionen [Int20f].

SGX SDK Das Software Development Kit (kurz: SDK) für Intel SGX wird nur auf Entwicklungsrechnern benötigt und stellt Bibliotheken bereit, die zum Erstellen von SGX-Anwendungen verwendet werden können (z.B. mit Kryptografiefunktionen).
Zusätzlich enthält das SDK verschiedene Hilfsprogramme, die bei der Entwicklung benötigt werden, u.a. zum Generieren von Bridge Functions und Signieren oder Debuggen von Enklaven. Außerdem sind diverse Vorlagen und Beispielcode enthalten [Ada18b; Geo19a].
Die Spezifikation der Programmierschnittstelle (engl.: Application Programming Interface, kurz: API) für das SGX SDK stellt Intel für Linux [Int20k] und Windows [Int20l] bereit.

2.1.6 SGX-Funktionen

Von den vielen sicherheitsrelevanten Funktionen und Möglichkeiten, die Secure Enclaves bieten, sollen die wichtigsten in diesem Abschnitt vorgestellt und erläutert werden. Sie sind als Voraussetzung für das Vertrauen in diese Trusted Execution Environment, und damit die Gesamtsicherheit von SGX, notwendig.

2.1.6.1 Enclave Attestation

Damit „trusted Hardware“ der Problemstellung von Secure Remote Computation begegnen kann, ist ein Vertrauen in die Funktionsweise der involvierten Hard- und Softwarekomponenten – und damit deren Hersteller – erforderlich. Im Fall von SGX umfassen diese den Prozessor (inkl. MEE) sowie alle genutzten Treiber und Softwarebibliotheken (die PSW inkl. der AEs). Zusätzlich muss ein Nutzer die Integrität des in SGX-Enklaven ausgeführten *tCodes* überprüfen können, um eine Manipulation schon vor dem Laden in die geschützte Enklave auszuschließen.
Um beides zu beweisen, kommt bei Intel SGX „(Software) Attestation“ zum Einsatz. Beim Attestieren einer Enklave gegenüber einer anderen auf der selben Plattform (der selben CPU) wird die „Local Attestation“ genutzt. Verwendet der Überprüfende dagegen einen anderen Rechner, ist stattdessen eine „Remote Attestation“ notwendig [Ana+13, S. 2].
Erst nachdem die Integrität durch diese Verfahren sichergestellt wurde, können vertrauliche Informationen (z.B. kryptografische Schlüssel) mit der entsprechenden Enklave ausgetauscht werden. Dieser Vorgang wird als „Secret Provisioning“ bezeichnet.

Local Attestation Die lokale Attestierung ermöglicht einer Enklave E_A, die Authentizität einer weiteren, auf dem selben physischen Prozessor ausgeführten, Enklave E_B – mit $CPU(E_A) \equiv CPU(E_B)$ – zu überprüfen. In diese Prüfung fließen die folgenden Informationen ein (nach [Ana+13, S. 2 f.] und [CD16, S. 84]):

- Identität der Enklave (`MRENCLAVE`)
- Identität des Autors der Enklave (`MRSIGNER`)
- Metadaten der Enklave (z.B. Version des *tCodes* oder verwendeter SGX-Modus)
- Nutzerdaten
- Identität der SGX-Plattform (u.a. `ProvisioningSecret` und `SealSecret`)

Zusammengefasst werden diese Daten in einer SGX-Datenstruktur, dem `REPORT`, wel-

cher spezifisch für eine zu attestierende (E_B) und eine verifizierende (E_A) Enklave erstellt wird, was nur E_A eine Überprüfung des `REPORTs` ermöglicht. Da die Identität der SGX-Plattform in die generierte Datenstruktur einfließt, kann der `REPORT` nur bestätigt werden, wenn sich beide Enklaven den selben Prozessor teilen.
Die optionalen Nutzerdaten werden beispielsweise verwendet, um einen Schlüsselaustausch (z.B. basierend auf dem Diffie-Hellman-Protokoll [DH76]) zwischen E_A und E_B durchzuführen, wobei die ausgetauschten Schlüssel nach erfolgreicher Attestation zur Absicherung weiterer Kommunikation genutzt werden [Geo19c].

Remote Attestation Soll eine Enklave auf einer anderen Plattform (z.B. der eines Cloud-Anbieters) attestiert werden, ist die Verwendung von gleichen Hardware-Secrets als Vertrauensgrundlage nicht möglich, sodass eine weitere Stelle als Vermittler benötigt wird. Da der Hersteller Intel die Integrität jeder SGX-Plattform mit Hilfe des gespeicherten `ProvisioningSecrets` validieren kann, tritt dieser als Vermittlungsstelle bei der Attestierung von Enklaven auf entfernten Rechnern auf.
Im Normalfall erfolgt die Verifikation des bei der Remote Attestation erstellten `QUOTEs` über das Internet deshalb durch den „Intel Attestation Service“ (kurz: IAS). Zur Nutzung von Offline-Rechnern (wie z.B. in Rechenzentren) bietet Intel als Alternative aber auch den Einsatz eigener Verifikationsserver mit Hilfe der „Data Center Attestation Primitives“ (kurz: DCAP) an [Sca+19].

Intel Attestation Service Bei der Authentizitätsprüfung einer auf entfernten Rechnern ausgeführten Enklave E_R durch einen Außenstehenden (*Challenger*) kommt die in Abschnitt 2.1.5 erwähnte *Quoting Enclave* zum Einsatz. Sie arbeitet intermediär zwischen der E_R und dem *Challenger* und attestiert Erstere auf der lokalen Plattform, um anschließend ein `QUOTE` zu generieren und an Letzteren zu schicken.
Zusätzlich ist aber eine Attestierung der QE durch den Hersteller notwendig, um die Authentizität der gesamten SGX-Umgebung zu verifizieren. Dieser, durch die Plattform-AEs unterstützte, Prozess wird „Provisioning“ genannt und verwendet das hinsichtlich des Datenschutzes optimierte Protokoll „Enhanced Privacy ID“ (kurz: EPID) [BL07] (siehe auch [Joh+16]).

Intel Data Center Attestation Primitives In bestimmten Szenarien ist es nicht möglich oder gewünscht, für eine Remote Attestation über das Internet auf den IAS-Dienst zuzugreifen. Als Alternative zur `QUOTE`-Verifikation durch Intel wurden 2018 die Data Center Attestation Primitives eingeführt, die den Einsatz eigener Offline-Verifikationsserver (nachfolgend als DCAP-VS bezeichnet) erlauben. Diese Funktion ist jedoch ausschließlich bei „Xeon E“-Prozessoren vorhanden [Joh18]. Das notwendige

Vertrauen in den Hersteller wird so allerdings nicht eliminiert, da vor der produktiven Nutzung von DCAP dennoch eine Kommunikation mit Intel-Servern erforderlich ist [Sca+19, S. 3 ff.].
Im Gegensatz zur bereits erläuterten Variante der Remote Attestation wird im DCAP-Szenario das von der QE generierte `QUOTE` nicht durch den IAS, sondern den DCAP-VS verifiziert. Dieser benötigt für den Abgleich eine eigene Datenbank mit Prozessorzertifikaten aller eingesetzten Plattformen, welche von Intel verifiziert und signiert werden müssen [Int20b]. Erst danach ist eine Abkopplung vom Internet möglich und ein *Challenger* kann die Authentizität der Enklave E_R durch den DCAP-VS überprüfen lassen [Sca+19, S. 3 ff.].
Zur Unterstützung dieses Attestation-Schemas müssen die folgenden Voraussetzungen erfüllt sein:

- Verwendung einer DCAP-fähigen CPU (der Serie Intel Xeon E) mit aktivierter „Flexible Launch Control"-Funktion[8] [Joh18]
- Einsatz der DCAP-Version des SGX-Treibers [Int20d]
- Nutzen einer speziellen (Intel- oder eigenen) DCAP-QE [Sca+19, S. 4 f.]

2.1.6.2 Sealing

Sollen die Daten einer Secure Enclave auch über deren Lebenszyklus hinaus verfügbar sein, ist eine kryptografisch „versiegelte" Persistierung vor Beendigung der Enklave erforderlich. Intel SGX nennt diese Datenspeicherung zum Austausch zwischen mehreren Enklaven (-instanzen) „Sealing".
Bei diesem Vorgang werden sensitive *eData* innerhalb der Enklave verschlüsselt und – mit Hilfe des $\mathcal{P}_u$ der A_{SGX} – auf einer Festplatte persistiert. Zur Verwendung der Daten in einer anderen Enklave erfolgt nach dem Lesen vom Massenspeicher eine Entschlüsselung mit dem selben Schlüssel („Unsealing"). Die Wahl des Verschlüsselungsverfahrens bleibt der Enklave überlassen, Intel empfiehlt jedoch zusätzlich zur eigentlichen Verschlüsselung den Einsatz von weiteren kryptografischen Primitiven wie *Nonces* zur Verhinderung von Replay-Angriffen [Ana+13, S. 4].

Für das Ver- sowie Entschlüsseln beim Sealing kommt ein `SealKey` zum Einsatz, welcher von beiden Hardware-Secrets abgeleitet wird und somit vom Hersteller nicht regeneriert werden kann. Der Entwickler beeinflusst die Erzeugung dieses Schlüssels –

[8] Flexible Launch Control ermöglicht den Austausch der von Intel signierten *Launch Enclave* durch eine eigene. Dadurch können Enklaven auch nach Regeln initialisiert werden, welche von denen der Original-LE abweichen. Z.B. wird die Verwendung einer speziellen, für DCAP angepassten, QE erlaubt [Int20g].

und damit die Eigenschaften des Sealings – durch Wahl einer „Key Policy“.
So kann der sichere Austausch von Daten nur zwischen mehreren Instanzen der selben Enklave erlaubt werden, indem der `SealKey` kryptografisch an die Enklavenidentität (`MRENCLAVE`) gebunden wird. Sollen *eData* dagegen über verschiedene Enklaven (bzw. unterschiedliche Versionen einer Enklave) hinweg ausgetauscht werden, kann stattdessen der `MRSIGNER`-Wert als Identität des Autors für die Schlüsselableitung verwendet werden.
Neben der Key Policy fließt auch die Versionsnummer der Enklave in den `SealKey` ein. Dadurch wird erreicht, dass nach einem sicherheitsrelevanten Update der A_{SGX} eine noch vorhandene Enklave der alten Version nicht auf neue *eData* zugreifen kann. Die Abwärtskompatibilität jedoch wird durch den Prozessor gewährleistet [Ana+13, S. 4 f., CD16, S. 81 ff.].

2.1.7 Einschränkungen und Erweiterungen

Die durch Intel SGX gesteigerte Sicherheit wird nur durch die Inkaufnahme verschiedener Einschränkungen in Bezug auf die Ausführungsgeschwindigkeit und Nutzungsfreundlichkeit erreicht. Einige Nachteile der ersten SGX-Generation konnten durch diverse Verbesserungen jedoch bereits abgeschwächt oder behoben werden.
In diesem Abschnitt sollen deshalb sowohl die wichtigsten technischen Beschränkungen als auch vorhandene Mitigationsmaßnahmen thematisiert werden. Neben den unvermeidbaren Performanzeinbußen aufgrund der durch den Prozessor durchgeführten Sicherheitschecks, des Wechsels zum und vom Enclave Mode sowie der Ver- und Entschlüsselung durch die MEE weisen SGX-Enklaven die folgenden Einschränkungen auf.

Langsames EPC-Paging Belegt eine Enklave zu viel Speicherplatz im EPC, müssen selten benutzte Speicherseiten in den unsicheren RAM ausgelagert werden. Um die Sicherheitseigenschaften von SGX zu wahren, erfolgt vor dem Paging eine Ver- und nach dem Zurückschreiben eine Entschlüsselung sowie verschiedene Maßnahmen zur Integritätssicherung durch CPU und MEE, was die Ausführungsgeschwindigkeit der Enklave verringert.
Durch die verdoppelte EPC-Kapazität seit der zehnten Generation von Intels „Core“-Prozessoren (siehe Abschnitt 2.1.4.1, Fußnote 5) steht Enklaven zwar mehr Speicherplatz zur Verfügung, dieser wird aber trotzdem unter allen auf dem Rechner laufenden Enklaven aufgeteilt. Die wirksamste Strategie zur Mitigation dieser Einschränkung bleibt also die Reduzierung der Enklaven-TCB.

Fehlende Systemcalls Dem Systemkernel wird im SGX-Szenario kein Vertrauen entgegengebracht, dieser befindet sich also außerhalb der TCB. Da bei einem Systemcall (kurz: Syscall) auf Kernelfunktionen zugegriffen wird und die Rückgabewerte von diesem manipuliert werden könnten, sind solche Funktionsaufrufe innerhalb einer Enklave verboten [Int20i, S. 35]. Dadurch fehlt dieser u.a. die Möglichkeit zum Zugriff auf I/O-Ressourcen [CD16, S. 90].

Wird die Funktionalität eines Systemaufrufs von einer Enklave benötigt, muss sie stattdessen einen OCall in ihren $\mathcal{P}_u$ und dieser den eigentlichen Systemcall durchführen. Zur Weiterleitung der Rückgabewerte ist schließlich wieder ein ECall erforderlich. Die beschriebene Prozedur führt – zusammen mit den Sicherheitsmaßnahmen beim Betreten und Verlassen einer Enklave – zu einer stark verringerten *tCode*-Geschwindigkeit.

Nachdem verschiedene Ansätze zu performanteren Systemcalls erforscht wurden (z.B. [Arn+16], [Ore+17] oder [WBA17]), führte Intel 2018 sog. „Switchless Calls“ als Alternative zu dem oben beschriebenen Verfahren im offiziellen SGX SDK ein.

Damit bleibt die Fähigkeit zum Aufrufen von Kernel-Code weiterhin dem $\mathcal{P}_u$ vorenthalten, die Enklave muss dafür allerdings nicht mehr verlassen werden. Stattdessen kommen asynchrone Anwendungsthreads zum Einsatz, welche den Systemcall durchführen und über einen zwischen $\mathcal{P}_u$ und $\mathcal{P}_t$ geteilten Speicherbereich mit dem *tCode* zur Übergabe von Funktionsparametern und -rückgabewerten kommunizieren. Durch die wegfallenden Schritte wird eine schnellere Enklavenausführung ermöglicht [Tia+18, S. 22 f.].

Statische Speicherallokation Bei der ersten SGX-Version muss der gesamte Speicherbedarf einer Enklave vor deren Initialisierung alloziert und alle Berechtigungen festgelegt werden, damit die betroffenen EPC-Seiten und deren Konfiguration in den `MRENCLAVE`-Hash aufgenommen werden können. Dadurch wird weder das nachträgliche Laden von Code oder Daten noch das Freigeben nicht (mehr) benötigten EPC-Speichers zur Nutzung durch andere Enklaven unterstützt.

Um diese Probleme zu lösen, wurden 2016 mit der zweiten Version von SGX zusätzliche CPU-Instruktionen zur Unterstützung von sicherer dynamischer Speicherverwaltung eingeführt [McK+16, S. 1 f.]

Internetzugriff für Remote Attestation Ursprünglich war für jede Remote Attestation die Verbindung zum Internet für die Kommunikation mit dem Intel Attestation Service notwendig.

Wie in Abschnitt 2.1.6.1 erläutert, erlaubt Intel durch die Einführung der Data Center Attestation Primitives nun jedoch auch den Einsatz eigener Offline-

Verifikationsserver, welche eine Attestation in einem geschlossenen Netzwerk von SGX-Rechnern ermöglichen.

Eine weitere Einschränkung, die SGX mit sich bringt, ist der fehlende Vertraulichkeitsschutz des *tCodes*. Da dieser durch das OS (mit CPU-Unterstützung) vom nicht vertrauenswürdigen Teil des Systems in die Enklave geladen wird, kann zwar seine Integrität (durch Überprüfung des `MRENCLAVE`-Wertes) sichergestellt werden, sein Inhalt bleibt vor einem möglichen Angreifer jedoch nicht verborgen (siehe Abschnitt 2.1.2). Diese Einschränkung kann nicht behoben, ihre Kritikalität jedoch reduziert werden, indem bei der Enklavenerstellung stattdessen ein intermediäres „Hilfprogramm“ in den EPC geladen (und dessen Integrität geprüft) wird. Nach Start der Enklave lädt es seinerseits den *tCode* nach und leitet jegliche ECall-Funktionsaufrufe an diesen weiter. Folglich können sowohl Enklavencode als auch -daten von einem vertrauenswürdigen Host bereitgestellt und verschlüsselt – und somit vertraulich – übertragen werden. Die Integritätsprüfung der *tCode*-Anweisungen muss in diesem Fall durch das Hilfsprogramm durchgeführt werden.

2.2 Migration von Nicht-SGX-Anwendungen

Bei der Entwicklung einer eigens auf die Nutzung im SGX-Umfeld ausgelegten Anwendung können die Besonderheiten von Secure Enclaves (u.a. Partitionierung in $\mathcal{P}_u$ und $\mathcal{P}_t$, Entwurf einer Kommunikationsschnittstelle und Verwaltung der Enklave) von Beginn an berücksichtigt werden (siehe Abschnitt 2.1.3).
Soll dagegen ein Teil oder die Gesamtheit des Codes bereits bestehender Softwarekomponenten mit möglichst geringer manueller Anpassung in einer SGX-Enklave ausgeführt werden, können diese SGX-Spezifika nicht durch die Anwendung selbst bereitgestellt werden. Stattdessen sind entweder (automatisierte) Codemanipulationen bzw. -erweiterungen erforderlich oder die benötigten Verwaltungsfunktionalitäten sowie Kommunikationsschnittstellen für SGX-Enklaven müssen von externen Diensten oder Werkzeugen bereitgestellt werden.
Um diese Aufgaben zu vereinfachen, wurden verschiedene wissenschaftliche Studien über unterschiedliche Ansätze bei der Migration bzw. Portierung einer klassischen Anwendung („Legacy Application“, kurz: A_L) in das SGX-Umfeld veröffentlicht, die in diesem Abschnitt vorgestellt und verglichen werden. Zusätzlich wird ein Überblick über verschiedene Privatunternehmen gegeben, welche Dienstleistungen oder Produkte für eine solche Migration bzw. die vereinfachte Verwendung von Intel SGX im Allgemeinen anbieten.

2.2.1 Migrationsstrategien

Die existierenden Ansätze setzen bei einer solchen Migration unterschiedliche Prioritäten. So verfolgen einige von ihnen beispielsweise das Ziel der vereinfachten Aufteilung einer Legacy Application in $\mathcal{P}_u$ und $\mathcal{P}_t$, während andere die komplette Anwendung in einer Enklave zur Ausführung bringen (eine solche, in eine Secure Enclave portierte, A_L wird nachfolgend als $A_{L,p}$ bezeichnet).
Nach dieser und weiteren Eigenschaften kann eine Kategorisierung der Studien über die Migration von Nicht-SGX-Anwendungen nach der zugrundeliegenden Migrationsstrategie mit absteigendem manuellen Aufwand wie folgt vorgenommen werden.

Programmierbibliothek Eine Bibliothek zur Programmierung von SGX-Anwendungen kann verwendet werden, um den vorhandenen Code der A_L um die für Enklaven notwendigen Funktionen zu erweitern bzw. zu modifizieren.
Ihre Verwendung bedeutet in der Regel den größten Aufwand bei der Migration, vor allem bei umfangreichen Anwendungen, weshalb sie sich (als alleinige Strategie) am wenigsten für diese Aufgabe eignen. Bei der Verwendung einer Alternative können Programmierbibliotheken jedoch ergänzend eingesetzt werden, um die Funktionalität, Performanz oder Sicherheit einer Enklave zu steigern [Int20k; Ore+17].

Automatisierungsframework Ein solches Framework kann zur Automatisierung verschiedener Aufgaben bei der Migration einer A_L eingesetzt werden, in der Regel erzeugt es sog. „Boilerplate Code“[9], welcher anderenfalls immer wieder manuell erstellt werden müsste.
Beispiele dafür sind die (teil-) automatisierte Partitionierung der Anwendung in *tCode* und $\mathcal{P}_u$-Code sowie die Generierung von Programmfragmenten für zusätzliche Funktionen wie verbesserte Systemcalls oder die Sicherheitsüberprüfung von Rückgabewerten [Tia+19; WBA17].

Shielding Layer Soll dagegen eine komplette Anwendung in einer Enklave ausgeführt werden, kann eine „Shielding Layer“ (dt.: Abschirmschicht) als Mittler zwischen dem Anwendungscode innerhalb und der OS-Schnittstelle außerhalb der Enklave zum Einsatz kommen. Diese Schicht fängt alle Funktionsaufrufe von innen nach außen (z.B. Systemcalls) ab und leitet sie an ihr Gegenstück im $\mathcal{P}_u$ zur Verarbeitung weiter, wobei sie eine Abschirmung des *tCodes* realisiert, z.B.

[9] „Boilerplate Code“ bezeichnet Teile im Quelltext, welche die folgenden vier Eigenschaften aufweisen: (1.) wenig Beitrag zur Funktionalität, (2.) häufige Verwendung, (3.) Lokalität und (4.) geringe Varianz zwischen mehreren Vorkommen [Nam19, S. 1253].

durch transparente Verschlüsselung von Netzwerkpaketen oder Integritätschecks von Rückgabewerten.
Im Gegensatz zum Ansatz des Library OS bewirkt die Verwendung einer Shielding Layer eine geringere Vergrößerung der TCB, gibt dafür aber eine umfangreiche Schnittstelle zum nicht vertrauenswürdigen Teil des Systems frei [Arn+16, S. 692].

Library OS Auf Bibliotheken basierende Betriebssysteme können genutzt werden, um mit wenig Aufwand eine A_L unmodifiziert und als Ganzes in einer SGX-Enklave auszuführen. Ein solches „Library OS“ (kurz: libOS) besteht dabei aus verschiedenen Softwarebibliotheken, die einen Teil der Funktionalitäten eines klassischen Betriebssystems nachbilden und diesen innerhalb einer Secure Enclave verfügbar machen.
Ähnlich wie bei der Shielding Layer wird eine Zwischenschicht in der Enklave benötigt, die Aufrufe externer Funktionen durch den Code der $A_{L,p}$ abfängt und bei Bedarf an den $\mathcal{P}_u$ weiterleitet. Im Gegensatz zur vorhergehenden Migrationsstrategie beinhaltet ein libOS allerdings viel mehr Funktionalität, um die Mehrheit der (System-) Funktionsaufrufe selbst zu bearbeiten und die Enklave nur selten verlassen zu müssen (beispielsweise für I/O-Zugriffe).
Dadurch wird die TCB sehr stark vergrößert, die Schnittstelle zum $\mathcal{P}_u$, und damit dem Host-OS, jedoch minimiert [Arn+16, S. 692; TPV17, S. 645 ff.].

Zusammengefasst versprechen demnach die Shielding Layer und das Library OS – mit unterschiedlicher Priorisierung der TCB-Größe bzw. der Schnittstelle zum Host-OS – die einfachste Portierung einer A_L, während Programmierbibliotheken und Frameworks zur Automatisierung manueller Aufgaben mehr Migrationsaufwand erfordern. In jedem Fall ist die Wahl der einzusetzenden Strategie jedoch abhängig von den Erfordernissen des spezifischen Anwendungsfalls, wobei auch eine Kombination verschiedener Ansätze weitere Möglichkeiten bieten kann.

2.2.2 Vorhandene Migrationsansätze

Zum Zeitpunkt der Erstellung dieser Arbeit steht eine Vielzahl von Studien zur (Unterstützung der) Migration von Legacy Applications zur Verfügung, von denen die wichtigsten in den Tabellen A.1 bis A.4 (Anhang A) ihren zugrundeliegenden Migrationsstrategien zugeordnet und einander gegenübergestellt werden.
Die dabei zum Vergleich herangezogenen Kriterien sollen im Folgenden erläutert werden.

Aufgabe Diese, nur beim Vergleich der Automatisierungsframeworks vorhandene, Spalte dient der Übersicht über die (wichtigste) manuelle Aufgabe bei der Portierung bzw. Ausführung einer Anwendung in der SGX-Enklave, welche durch Verwendung des jeweiligen Ansatzes (teil-) automatisiert werden kann.

TCB Wie im vorangehenden Abschnitt diskutiert, sollte aus Sicherheits- und Performanzgründen der Speicherbedarf einer Enklave so gering wie möglich gehalten werden. Deshalb ist es wichtig, dass die TCB der $A_{L,p}$ durch die Migration möglichst wenig erhöht wird.

Als Anhaltspunkt für den Einfluss auf den EPC-Speicherbedarf werden in den Tabellen die Anzahl der Codezeilen (engl.: Lines of Code, kurz: LOC bzw. kLOC für 1000 LOC) aufgeführt, welche dem eigentlichen Anwendungscode durch die Migration hinzugefügt werden.

Open Source Je nach Verwendungszweck ist es relevant, ob der Code des Migrationsansatzes öffentlich verfügbar ist und eingesehen werden kann. Da bei der Verwendung im Produktiveinsatz eine langfristige Unterstützung erforderlich ist, sollte außerdem darauf geachtet werden, dass dieser aktiv weiterentwickelt und gepflegt wird.

Aufwand Dieses Kriterium soll eine qualitative Schätzung abgeben, mit wie viel Aufwand eine Migration nach dem entsprechenden Ansatz verbunden ist. Dazu werden Einschätzungen von „sehr gering“ (nahezu automatische Migration mit wenigen manuellen Eingriffen) bis „sehr hoch“ (viele manuelle Schritte mit erheblichem fachlichen und zeitlichen Aufwand) getroffen.

Systemcall-Strategie Im Gegensatz zu dem in Abschnitt 2.1.7 beschriebenen Standard-Vorgehen zur Ausführung von Systemcalls aus einer Enklave setzen die betrachteten Portierungsmöglichkeiten teilweise andere Strategien ein, um eine verbesserte Sicherheit oder schnellere Ausführung zu erreichen.

Systemcall-Kompatibilität Je nach eingesetzter Syscall-Strategie und verwendeten Bibliotheken werden bei verschiedenen Migrationsansätzen nicht alle unter Linux verfügbaren Systemcalls unterstützt. Dadurch können bei einigen Legacy Applications unter Umständen Kompatibilitätsprobleme auftreten, welche eine Behebung durch manuelle Modifikation erfordern.

In dieser Spalte wird (wenn möglich) außerdem die jeweils für Syscall-Funktionen verwendete Standard-C-Bibliothek aufgeführt, wobei es sich entweder um die umfangreiche und z.Zt. populärste „GNU C Bibliothek“ (kurz: *glibc*) [Fre20] oder die auf geringen Speicherbedarf optimierte „musl libc“ [McC20] mit teils abweichender Funktionalität handelt.

Einschränkungen Verändert ein Migrationsansatz bestimmte Eigenschaften oder Vorgehensweisen von SGX, können in der Nutzung verschiedener Funktionen Einschränkungen für die Anwendung auftreten. Einige Lösungen stellen auch besondere Anforderungen an ihre Umgebung oder die zu portierende Anwendung.

Zusammenfassung Da die im Anhang A vorgestellten Studien jeweils unterschiedliche Schwerpunkte setzen, ist die Erstellung einer eindeutigen Rangliste über die Eignung zur Migration von Legacy Applications nicht möglich. Stattdessen muss im jeweiligen Anwendungsfall geprüft werden, welche Anforderungen an die $A_{L,p}$ bestehen und mit welchen Ansätzen diese möglichst effizient erreicht werden.

Handelt es sich bei der zu portierenden Anwendung um eine relativ kleine und wenig komplexe Fremd- oder eine Eigenentwicklung, kann z.B. eine manuelle Migration mit Hilfe des SGX SDK [Int20k] (und dem Einsatz von „Switchless Calls“ [Tia+18] zur Performanzsteigerung) oder eine Anpassung des Quellcodes zur Nutzung von „Eleos“ [Ore+17] praktikabel sein und zu sehr guten Ergebnissen (in Bezug auf die Minimierung der TCB und hohe Performanz) führen.
Ist ein solcher Programmieraufwand nicht erwünscht, können Frameworks wie „Glamdring“ [Lin+17] oder „lxcsgx“ [Tia+19] zur teilautomatisierten Partitionierung der Anwendung in $\mathcal{P}_u$ und $\mathcal{P}_t$ verwendet werden. Auch wenn dafür keine manuelle Modifikation des Anwendungsquellcodes erforderlich ist, wird eine gewisse Kenntnis dessen vorausgesetzt – z.B. müssen jene Variablen identifiziert werden, die schützenswerte Daten enthalten können und deshalb nur in einer Enklave verarbeitet werden dürfen. Soll die Performanz von der $A_{L,p}$ ausgeführter Systemcalls verbessert werden, kann eine teilautomatisierte Migration der gesamten Anwendung durch das Framework „HotCalls“ [WBA17] in Betracht gezogen werden. Das Hauptaugenmerk der Lösung „Panoply“ liegt dagegen auf einer Minimierung der TCB und funktionalen Partitionierung der Anwendung in mehrere kleine Enklaven. Mit Hilfe von Quellcodeannotationen durch den Nutzer soll das Framework die Gesamtperformanz und Sicherheit der Applikation steigern.
Ist die Anwendung zu komplex und sind keine weitreichenden Kenntnisse über deren Interna vorhanden (z.B. im Fall eines Open-Source-Webservers), kann mittels einer Shielding Layer wie „SCONE“ [Arn+16] oder „ScriptShield“ [Wan+19] eine Migration der gesamten A_L mit vergleichsweise geringem manuellem Aufwand und zusätzlichen Sicherheitsfunktionen erreicht werden.
Den grundsätzlich geringsten Zeitaufwand bei der Migration bringt ein Library OS mit sich, da die Ausführung der Anwendung jener im „normalen“ Betriebssystem sehr ähnlich ist. Die verschiedenen diese Strategie verfolgenden Ansätze setzen dabei

unterschiedliche Schwerpunkte. So zielt z.B. „SGX-LKL“ [Pri+20] auf eine kleinstmögliche Schnittstelle zum Host-OS ab, während „SGXKernel“ [Tia+17] die Minimierung der Enklaven-TCB fokussiert. Die Unterstützung von Multitasking und -threading innerhalb der Enklave ist dagegen ein Schwerpunkt von „Occlum“ [She+20]. Je nach Priorisierung dieser Interessen in einem spezifischen Migrationsfall müssen die verschiedenen Lösungen gegeneinander abgewogen werden.
Hierbei ist anzumerken, dass das libOS „Haven“ [BPH15] das einzige für die Ausführung von Windows-Anwendungen geeignete ist, während alle anderen ein Linux-System voraussetzen.

2.2.3 Proprietäre Lösungen

Neben den, in Anhang A gegenübergestellten, wissenschaftlichen Studien werden auch von diversen Privatunternehmen Dienstleistungen bzw. Lösungen zur Erleichterung der Migration von Nicht-SGX-Anwendungen angeboten. Da im kommerziellen Umfeld ein minimaler zeitlicher und finanzieller Aufwand von besonderer Relevanz ist, wird dabei häufig eine besonders einfache und schnelle Migration versprochen.
Zur Erweiterung der Übersicht über mögliche Werkzeuge werden in diesem Abschnitt die wichtigsten dieser proprietären Ansätze zur Migration bzw. Ausführung in SGX kurz vorgestellt.

In diesem Zusammenhang ist das von der Linux Foundation ins Leben gerufene „Confidential Computing Consortium“ (kurz: CCC) von besonderer Relevanz. Dieser Zusammenschluss zahlreicher Hardware-, Software- und Dienstleistungsunternehmen hat das Ziel, den Austausch und die Zusammenarbeit der Mitglieder zu fördern, um die Weiterentwicklung und den Einsatz von Trusted Execution Environments wie Intel SGX zu beschleunigen [Con20c].
Da deshalb viele wichtige in diesem Themenbereich tätige Firmen Teil des CCC sind, wird dieses als Grundlage für die nachfolgende Auflistung verwendet. Während die komplette Mitgliederliste unter [Con20b] abgerufen werden kann, soll an dieser Stelle eine Auswahl der für die Migration einer A_L relevanten Mitglieder und deren Angebote vorgestellt werden.

Prozessorhersteller Intel, AMD und ARM treten als Anbieter der für TEE notwendigen Hardware auf. Neben Intel mit den dieser Arbeit zugrundeliegenden Software Guard Extensions bieten auch die anderen beiden Unternehmen eine Implementierung des TEE-Prinzips – mit von SGX abweichenden Eigenschaften – an [KPW16; Nga+16]. Für eine Gegenüberstellung von Intel SGX und „AMD

Memory Encryption“ kann z.B. [Mof+18] herangezogen werden.

Public-Cloud-Anbieter Unternehmen wie Google oder Microsoft erweitern ihr Angebot für Cloud-Computing mehr und mehr um Funktionen für vertrauenswürdige Rechenressourcen unter Nutzung von AMD- oder Intel-Technologien. So erlaubt Microsoft in seiner „Azure“-Cloud beispielsweise die Verwendung von virtuellen Maschinen oder Container-Clustern mit zugrundeliegenden SGX-Prozessoren [Rus18].
Weiterhin entwickelt Google mit „Asylo“ ein Open-Source-Framework zur Nutzung von TEE-Funktionalität mit Abstraktion der spezifischen Implementierung (z.B. Intel SGX) und der Möglichkeit zur Portierung von bestehenden Anwendungen [Asy20].

Ant Group Das zum chinesischen Konzern Alibaba Group gehörende Unternehmen arbeitet an der Kombination von SGX-Enklaven mit Containern und entwickelt dazu „Inclavare Containers“, eine Laufzeitumgebung für in Containern gekapselte SGX-Anwendungen. Zur Ausführung von bestehenden Applikationen kommt dazu eine der beiden libOS-Implementierungen Graphene-SGX oder Occlum zum Einsatz [Inc20].

Anjuna Das US-amerikanische Unternehmen bietet eine einfache Migration von bestehenden Anwendungen in das Intel-SGX-Umfeld sowie die Nutzung verschiedener, bereits portierter Applikationen an [Anj20].

CYSEC Unter Nutzung der Software Guard Extensions entwickelt CYSEC den eigenen Migrationsansatz „ARCA“, der Kundenanwendungen in sicheren Enklaven ausführt. Zusätzlich stellt auch dieser Anbieter diverse bereits migrierte und einsatzbereite Anwendungen bereit [CYS20].

Edgeless Systems Dieses deutsche Unternehmen bietet ein in der „Community Version“ als Open Source vorliegendes SDK für Intel SGX an, welches u.a. die Migration von bestehenden Anwendungen vereinfachen soll. Zusätzlich werden verschiedene Softwarelösungen bereitgestellt, die unter Nutzung dieses SDK entwickelt wurden [Sch20].

Fortanix Diese Firma stellt Lösungen und Dienstleistungen zur Migration von Nicht-SGX-Anwendungen ohne die Notwendigkeit von Codemodifikationen mittels eines eigenen Library OS bereit. Außerdem werden verschiedene weitere Dienste im Zusammenhang mit SGX-Anwendungen angeboten, z.B. durch Enklaven geschützte Schlüsselverwaltung oder eine Software zur zentralisierten Verwaltung und Attestierung von Enklaven [For20].

MADANA Das deutsche Unternehmen bietet verschiedene Dienste zur Absicherung von Anwendungen mittels Intel-SGX-Enklaven an, die nach dem Prinzip eines auf „Alpine Linux“ basierenden Library OS arbeiten. Neben kommerzieller Unterstützung werden zudem Verwaltungswerkzeuge und eine Platform-as-a-Service-Umgebung (kurz: PaaS) für SGX-Anwendungen bereitgestellt [MAD20]

Neben den kommerziellen Mitgliedern umfasst das CCC verschiedene Open-Source-Projekte aus dem Themengebiet der Trusted Execution Environments. Unter den Projekten, welche sich noch in der Beitrittsphase befinden, sind u.a. auch die bereits vorgestellten Library-OS-Implementierungen Graphene-SGX und Occlum enthalten [Con20a].

2.2.4 Betrachtungen zur Anwendungspartitionierung

Wie in Abschnitt 2.1.3 erläutert, hat der Hersteller Intel SGX mit dem Ziel entworfen, eine geschützte Umgebung für einen kleinen, abgegrenzten Teil einer Anwendung bereitzustellen, welcher sensitive Daten verarbeitet [Int20j]. Die Ausführung von kompletten Anwendungen in einer Enklave gehört dagegen nicht zu den Stärken der Befehlssatzerweiterung. Aus diesem Grund bietet der ursprüngliche Enclave Page Cache nur einen geringen Speicherplatz.
Folglich kann z.B. die beste Performanz erreicht werden, wenn eine bestehende Anwendung für den Einsatz in SGX umfassend manuell modifiziert (bzw. neu entwickelt) wird. Zur Reduktion des dafür notwendigen Aufwands kann z.B. ein Framework wie „Glamdring“ [Lin+17] verwendet werden, wobei sensitive Daten verarbeitende Codefragmente lediglich annotiert werden müssen (siehe Abschnitt 2.2.2). Eine Kenntnis des Quellcodes ist jedoch in jedem Fall erforderlich.
Die Minimierung der Enklaven-TCB hat nicht nur Vorteile für die Ausführungsgeschwindigkeit der gesamten A_{SGX}, sie trägt auch zur zusätzlichen Steigerung der Sicherheit bei. Selbst wenn eine solche Anwendung vor unerlaubten Zugriffen durch das Betriebssystem oder einen privilegierten Angreifer geschützt wird, hängt die Gesamtsicherheit immer von den ausgeführten Anweisungen des *tCodes* ab. Dementsprechend ist die Wahrscheinlichkeit einer darin enthaltenen Schwachstelle umso größer, je mehr Anwendungsfunktionalität (Programmcode oder Bibliotheken) sich innerhalb einer Enklave befindet.

Ein Praxisbeispiel für diesen Zusammenhang ist der sog. „Heartbleed-Bug“, welcher 2014 in der TLS-Bibliothek OpenSSL gefunden wurde [Syn20]. Dabei ist eine in dieser enthaltene Funktion für „Heartbeat“-Pakete betroffen, welche die vorzeitige Terminie-

rung einer aktiven TLS-Verbindung verhindern sollen und über die unautorisierter Zugriff auf sensitive Informationen erlangt werden kann.
Wird eine OpenSSL verwendende Anwendung in ihrer Gesamtheit migriert, befindet sich auch diese Bibliothek komplett im $\mathcal{P}_t$ und der Heartbeat-Code besitzt Zugriff auf alle vertraulichen Daten. Folglich kann ein Angreifer durch Ausnutzung dieses Programmierfehlers solche Daten erlangen, obwohl die Anwendung durch Intel SGX geschützt wird.
Da die Existenz bisher unbekannter Schwachstellen bei keiner Software ausgeschlossen werden kann, sollte sich also nur jene Funktionalität in einer Enklave befinden – und damit Zugriff auf deren schützenswerte Daten besitzen –, für die das zwingend erforderlich ist. In diesem Beispiel könnte durch Partitionierung der Kryptografie-Bibliothek der für Heartbeats verantwortliche Code im $\mathcal{P}_u$ ausgeführt werden, während der $\mathcal{P}_t$ lediglich die Bibliotheksfunktionen für die eigentliche Verschlüsselung und die Schlüsselverwaltung umfasst. Damit hätte der Heartbleed-Bug die Vertraulichkeit und Integrität der sensitiven Daten nicht gefährden können.

Fazit Obwohl eine restriktive Partitionierung folglich sowohl für die Performanz als auch die Sicherheit der betroffenen Anwendung den besten Weg darstellt, ist dieser Ansatz für die Migration einer bestehenden Applikation oft nicht praktikabel.
Der Nutzer einer Open-Source-Dokumentenverwaltung kann z.B. in der Regel keine Kenntnis des Quellcodes vorweisen und ist demnach (ohne umfassendes Studium dessen) nicht in der Lage, jene Variablen und Funktionen zu detektieren, welche sensitive Daten speichern oder verarbeiten. Der manuelle Aufwand steht dabei im Verhältnis zur Komplexität (und damit Umfang des Programmcodes) der Anwendung. Im Fall von Closed-Source-Software besteht (ohne Mithilfe des Entwicklers) gar keine Möglichkeit zur Änderung des Quelltextes.
Aus diesem Grund stellt die Minimierung der Enklaven-TCB durch Partitionierung zwar die optimale Herangehensweise dar, kann in der Praxis allerdings oft nicht durchgeführt werden. Stattdessen wird zur Verringerung des Migrationsaufwandes eine geringere Ausführungsgeschwindigkeit bzw. Sicherheit in Kauf genommen.

Vergrößerung des EPC Eine aktuelle Entwicklung von Seiten des Herstellers weist voraussichtlich einen großen Einfluss auf die beschriebenen Erwägungen auf. So stellte Intel im Oktober 2020 eine neue Generation des Server-Prozessors „Xeon“ vor und kündigte dabei eine starke Vergrößerung des durch Enklaven nutzbaren Speichers auf bis zu einem TB an [Int20c].
Unter diesen Umständen verliert die Minimierung der Enklaven-TCB an Bedeutung, da die Auslagerung von EPC-Seiten – und die damit verursachte Performanzreduktion

– viel seltener erfordert (und in der Praxis bei vielen Anwendungen gänzlich eliminiert) wird. Werden diese Ankündigungen umgesetzt, sind u.a. die folgenden Tendenzen zu erwarten.

- Jede A_{SGX}, welche mehr als den aktuellen EPC-Speicher benötigt, kann schneller ausgeführt werden.
- Die Partitionierung bestehender Anwendungen weist keinen Performanzvorteil gegenüber einer kompletten Migration (z.B. durch ein libOS) auf.
- Auf einem physischen Rechner kann eine größere Anzahl von Enklaven – und damit mehr SGX-Anwendungen – gleichzeitig ausgeführt werden.

Programmierbibliotheken oder Partitionierungswerkzeuge können folglich den mit ihrer Nutzung verbundenen Aufwand durch weniger Vorteile aufwiegen. Die in diesem Abschnitt beschriebene gesteigerte Sicherheit aufgrund der geringeren Wahrscheinlichkeit für Programmierfehler und Schwachstellen stellt allerdings trotzdem einen validen Grund für die Nutzung dieser Migrationsstrategien dar.

3 Migration von Multi Service Applications

Viele der in Abschnitt 2.2 aufgeführten Studien zur Migration von Nicht-SGX-Anwendungen gehen davon aus, dass diese Applikationen als eine Einheit aus der entsprechenden Programmdatei und den benötigten Bibliotheken vorliegen. In der Praxis wird eine solche Annahme jedoch oft nicht erfüllt.
Vielfach kommen hierbei stattdessen komplexere, „serviceorientierte“ Anwendungen zum Einsatz, welche verschiedene zusammenhängende Aufgaben erfüllen und sich aus mehreren einzelnen Diensten (kurz: svc_i) zusammensetzen. Diese verfügen ihrerseits über definierte Schnittstellen und Kommunikationskanäle, um miteinander und mit ihrer Umwelt in Interaktion zu treten (kurz: $C(\text{svc}_i, \text{svc}_j) : i \neq j$ bzw. $C(\text{I/O}, \text{svc}_i)$). Das zugrundeliegende Prinzip wird als „Service Oriented Architecture“ [ISO16] bzw. „Microservice-Prinzip“[1] und eine solche Anwendung im Folgenden als „Multi Service Application“ (kurz: A_{MS}) bezeichnet.
Eine Veranschaulichung deren strukturellen Aufbaus erfolgt in Abbildung 3.1.

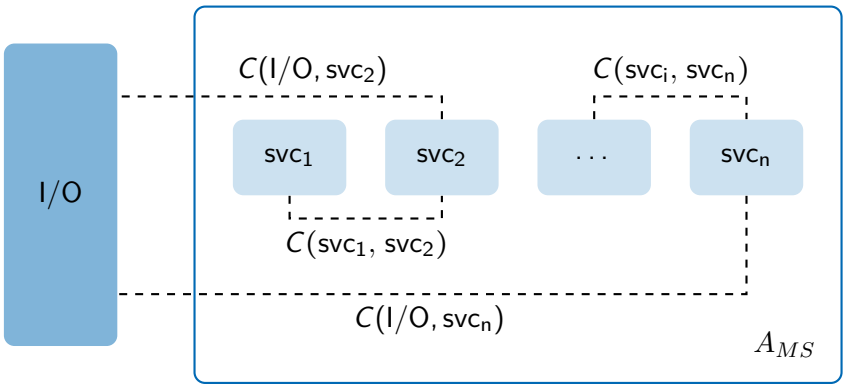


Abbildung 3.1: Aufbau einer Multi Service Application

Soll eine solche Anwendung mittels Intel-SGX-Enklaven geschützt werden, ist eine Migration mit mehreren Schritten erforderlich, welche im Folgenden erläutert werden, wobei zur Illustration des Vorgehens das Beispiel einer Webanwendung (kurz: WebApp) herangezogen wird.

[1] Bei dieser Architektur wird zur Komplexitätsreduktion die Aufteilung von monolithischen Applikationen in mehrere kleine, voneinander möglichst unabhängige Teile vorgesehen, die über definierte Schnittstellen kommunizieren [Nad+16, S. 6 f.].

Solche Anwendungen stellen typische Vertreter von A_{MS} dar und machen – getrieben vom Trend des interaktiven „Web 2.0“ [Alb08] – einen immer größeren Teil der verfügbaren Webseiten im Internet aus. Das Grundprinzip ist hierbei, dass die Funktionalität einer Software nicht in Form einer auf dem lokalen Rechner ausführbaren Binärdatei, sondern bedarfsgerecht durch das Zusammenspiel von Client- und Server-Anwendungen über das Internet bereitgestellt wird [SP08].

Der gesamte Prozess einer A_{MS}-Migration in das Umfeld von Intel SGX führt zur portierten Anwendung $A_{MS,p}$ und kann in vier, in den nachfolgenden Abschnitten genauer erläuterte, Schritte untergliedert werden:

1. Analyse der Anwendungsstruktur sowie der einzelnen svc_i
2. Entwurf der anwendungsspezifischen Enklavenarchitektur
3. Auswahl der eingesetzten Migrationsstrategie(n) und -ansätze sowie weiterer Hilfsmittel
4. Implementierung der geplanten Migration

3.1 Analyse der Anwendungsstruktur

Um die für die Migration einer bestimmten Anwendung optimale Architektur der SGX-Umgebung sowie die passenden Migrationsansätze auszuwählen, muss die A_{MS} zuerst strukturell analysiert und ihre für SGX relevanten Eigenschaften herausgestellt werden.
Das Ziel dieses Schrittes ist die Identifikation der schützenswerten Daten als Grundlage, der einzelnen Dienste, aus denen sich die A_{MS} zusammensetzt, sowie deren Kommunikationsschnittstellen. Zusätzlich müssen die jeweiligen technischen Spezifika und Anforderungen bestimmt werden.

Kritische Daten Als Erstes ist es erforderlich, die Art und den Umfang jener besonders kritischen (bzw. sensitiven) Daten der spezifischen Anwendung zu definieren, welche durch den Einsatz der Software Guard Extensions vor Angriffen auf die Vertraulichkeit und Integrität geschützt werden sollen. Diese Definition stellt die Voraussetzung für alle weiteren Schritte dar, die sich stets an dem übergeordneten Ziel der Migration – dem Schutz der kritischen Daten – orientieren.

Beispiel: Eine Webanwendung kann beispielsweise persönliche Informationen verschiedener Nutzer verarbeiten, vertrauliche Dokumente verwalten und den Zugriff auf einige Daten nur nach erfolgreicher Anmeldung und Authentifizierung gestatten.
Sobald sie nicht ausschließlich öffentlich einsehbare Inhalte bereitstellt, wird ein Vertraulichkeitsschutz benötigt. Das Verhindern von Angriffen auf die Integrität der verarbeiteten Daten ist sogar in jedem Fall notwendig, damit z.B. ein unberechtigter Nutzer keine Falschinformationen verbreiten kann.

Dienste Zur Identifikation der svc_i einer Multi Service Application kann in der Regel die technische Dokumentation herangezogen werden, wobei unter Umständen – insbesondere bei konsequenter Umsetzung des „Microservice-Prinzips“ und ausreichender Unabhängigkeit der einzelnen Dienste untereinander – mehrere alternative Zusammensetzungen (in Bezug auf Diensttypen oder -implementierungen) bestehen. In solch einem Fall müssen alle Möglichkeiten berücksichtigt werden, um bei Bedarf eine abweichende Dienstauswahl für die $A_{MS,p}$ zu treffen.
So benötigen viele größere Anwendungen als persistenten Speicher beispielsweise eine Datenbank. So lange das zugrundeliegende Datenbankmodell dem der auf die Daten zugreifenden Dienste entspricht, spielt die Wahl der verwendeten Dienstimplementierung bei grundlegender Unterstützung eine untergeordnete Rolle. Beispielsweise unterstützen sowohl „MariaDB“ [Mar20c] als auch die „Oracle Database“ [Ora20] das relationale Datenbankmodell.
An dieser Stelle muss unterschieden werden zwischen Diensten, die in kompilierten Sprachen wie C oder Rust geschrieben sind, und interpretierten Programmen (z.B. PHP-, Python- oder Lua-Skripten).
Erstere werden in der Regel als ausführbare Datei bereitgestellt und können auf dem Zielsystem direkt ausgeführt werden. Zur Verwendung in Intel SGX muss deshalb lediglich diese Programmdatei (und evtl. erforderliche Bibliotheken) portiert werden.
Interpretierte Anwendungen dagegen werden oft als Projekt von Quellcode-Dateien bereitgestellt und benötigen für jeden Programmstart einen Interpreter, der diesen verarbeitet und die entsprechenden Befehle ausführt. Deshalb ist in diesem Fall zuerst die Migration des sprachenspezifischen Interpreters notwendig, der dann beliebige Skriptdateien zur Ausführung bringen kann.
Die zuvor entwickelte Definition kritischer Daten wird verwendet, um jene svc_i auszumachen, die auf solche Informationen zugreifen und folglich besonderen Schutzes durch SGX-Enklaven bedürfen. Unter Umständen kann sich dabei herausstellen, dass einige Dienste der Anwendung auf die Sicherheit der kritischen Daten keinen Einfluss haben – und demnach nicht migriert werden müssen.

Nach der bloßen Identifikation der relevanten A_{MS}-Dienste gilt es wiederum in Er-

fahrung zu bringen, wie diese sich zusammensetzen, welche ausführbaren Dateien sie demnach verwenden und welche (statischen oder dynamischen) Bibliotheken, Module oder Erweiterungen zusätzlich benötigt werden.
Bei der Auflistung dynamischer Bibliotheken müssen nicht nur jene Beachtung finden, welche während des „Linking"-Prozesses vom dynamischen Linker in der Anwendung eingetragen werden. Auch solche, die evtl. während der Programmlaufzeit durch die Anwendung selbst nachgeladen werden[2], müssen berücksichtigt werden. Da in einer Secure Enclave nur vertrauenswürdiger *tCode* nach einer Integritätsprüfung ausgeführt werden darf, ist es erforderlich, dass alle möglicherweise verwendeten Bibliotheken vor der Enklavenerstellung bekannt sind.

Beispiel: Die Struktur einer Webanwendung folgt in der Regel der sogenannten „3-Tiered Architecture", sie besteht also aus drei Schichten mit distinkten Aufgaben. Das *Presentation Tier* ist für die Darstellung der angeforderten Inhalte sowie jegliche Kommunikation mit dem Client zuständig, während das *Application Tier* die eigentliche Applikationslogik enthält und Berechnungen auf den Daten durchführt. Das *Storage/Database Tier* dient schließlich der persistenten Speicherung der Anwendungsdaten (z.B. in einer Datenbank oder einzelnen Dateien) sowie deren Bereitstellung für Dienste im *Application Tier* [LHS05, S. 1 f.].
Bei vielen einfachen Webanwendungen wird für jede Schicht ein einzelner Dienst verwendet, was zum folgenden Aufbau führt:

Tier 1 (*Presentation*): Webserver (kurz: WS)

Tier 2 (*Application*): Anwendungsserver (kurz: AS)

Tier 3 (*Storage*): Datenbank bzw. -server (kurz: DB)

Alle drei Dienste können Zugriff auf schützenswerte Daten der WebApp nehmen. Der Webserver terminiert die bei der Verbindung zum Nutzer eingesetzte Verschlüsselung und verarbeitet so alle dem Anwender zugänglichen Informationen. Der Anwendungsserver benötigt kritische Daten als Ein- und Ausgabewerte und die Datenbank speichert und organisiert jegliche persistente Informationen. Demzufolge müssen alle diese Dienste durch SGX-Enklaven geschützt werden.

Schnittstellen Wurden die vorhandenen Dienste identifiziert, muss anschließend analysiert werden, in welcher Zusammenstellung und über welche Schnittstellen sie miteinander sowie mit ihrer Umwelt kommunizieren und interagieren.

[2]Unter Linux wird zum Nachladen von Bibliotheken zur Laufzeit z.B. die `dlopen()`-Funktion verwendet [RQK15].

Außerdem müssen weiterführend die verfügbaren Kommunikationsmittel und -protokolle sowie die jeweils unterstützten Sicherheitsmaßnahmen (z.B. Verschlüsselungsverfahren) jeder Schnittstelle bestimmt werden, um einen sicheren Datenaustausch zwischen den Anwendungsdiensten – und später den Enklaven der $A_{MS,p}$ – gewährleisten zu können. Auch dabei ist es wichtig, alle möglichen Alternativen zu berücksichtigen, um jeweils die für das spezifische SGX-Szenario optimalen auszuwählen.
In einer Linux-Umgebung kann beispielsweise die Wahl zwischen einem TCP- [KK08] oder Unix Domain Socket [KKS16] zur lokalen Interprozesskommunikation (engl.: Inter-Process Communication, kurz: IPC) bestehen, wobei der Informationsaustausch in ersterem Fall etwa durch eine Verschlüsselung auf Basis des TLS-Protokolls [Res18] abgesichert werden könnte.

Beispiel: Im erläuterten Szenario einer WebApp nach der „3-Tiered Architecture“ wird der Webserver verwendet, um Anfragen vom Client (z.B. dem Webbrowser eines Applikationsnutzers) entgegenzunehmen und letztlich zu beantworten. Betrifft eine solche Anfrage ausschließlich statischen Inhalt (z.B. eine HTML- oder CSS-Datei), kann diese vom WS ohne die Einbeziehung weiterer Dienste beantwortet werden.
Bei der Anforderung dynamischer Daten müssen die ausgelieferten HTML-Inhalte dagegen noch generiert werden, wofür der Anwendungsserver zuerst die erforderlichen Berechnungen der Applikationslogik durchführt und die Ergebnisse anschließend an den WS weiterreicht.
Wenn bei solchen Berechnungen schließlich in einer Datenbank gespeicherte Daten als Eingabe benötigt oder als Ergebnis erzeugt werden, stellt der AS seinerseits eine Anfrage an den DB-Server, um eine Abfrage oder Aktualisierung der erforderlichen Teilmenge des Datenbankinhaltes durchzuführen.
Dieser greift zu deren Beantwortung auf seine zugrundeliegende Datenbasis (engl.: Backing Storage, kurz: BSt) zu, welche z.B. aus einer Menge von Binärdateien auf einem Datenträger besteht.
Entsprechend dieser Ausführungen verwendet die analysierte A_{MS} also folgende Kommunikationskanäle:

C(Cl., WS): Client $\longleftrightarrow$ Webserver

C(WS, AS): WS $\longleftrightarrow$ Anwendungsserver

C(AS, DB): AS $\longleftrightarrow$ Datenbank

C(DB, BSt): DB $\longleftrightarrow$ Backing Storage

Zur Absicherung des Datenaustausches bestehen bei jedem dieser Kommunikationswege unterschiedliche Möglichkeiten. So werden die Nutzeranfragen und entsprechenden Antworten des Webservers in der Regel durch die Nutzung des HTTPS-Protokolls anstelle von Klartext-HTTP-Paketen verschlüsselt, wobei die Authentifizierung und der Schlüsselaustausch z.B. mit Hilfe von X.509-Zertifikaten [Coo+08] realisiert wird. Die für die Kommunikation C(WS, AS) sowie C(AS, DB) eingesetzten Protokolle hängen vorwiegend von der spezifischen Implementierung des Anwendungs- bzw. Datenbankservers ab. Dabei verwenden Erstere oft eine Variante der CGI-Spezifikation[3], während die meisten Datenbankmanagementsysteme ihr eigenes, datenbankspezifisches Kommunikationsprotokoll, wie das MySQL-Protokoll, mitbringen. Hier müssen durch Konsultation der jeweiligen Dokumentation verfügbare Sicherheits- bzw. Verschlüsselungsverfahren ermittelt werden.
Der Zugriff einer Datenbank auf ihre Datenbasis hängt ebenfalls von der verwendeten Software ab. Oft werden hierbei allerdings auf einem (logischen oder physischen) Speichergerät persistierte Dateien verwendet. Lese- und Schreibvorgänge auf diese erfolgen in der Regel über Schnittstellen des OS-Kernels, ggf. weitere Abstrahierungsschichten und das physische I/O-Gerät.

Nachdem die Analyse fertiggestellt wurde, kann die Struktur einer aus drei Diensten und vier Kommunikationskanälen bestehenden Webanwendung also wie in Abbildung 3.2 dargestellt zusammengefasst werden.

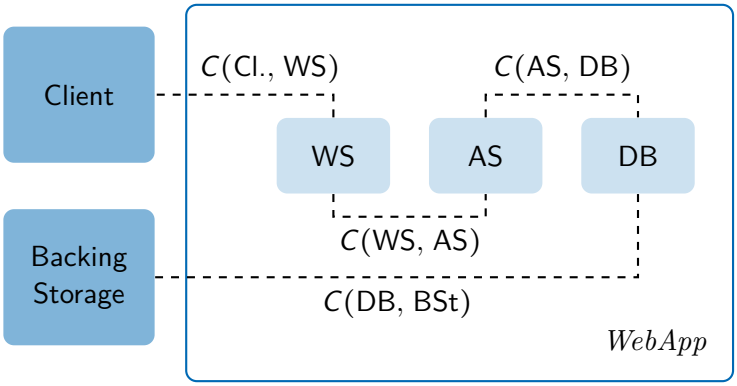


Abbildung 3.2: Aufbau einer beispielhaften Webanwendung

[3]Während das ursprüngliche Common Gateway Interface (kurz: CGI) [RC04] mittlerweile nur noch selten zum Einsatz kommt, werden heutzutage vorrangig Weiterentwicklungen wie FastCGI [Bro96] oder SimpleCGI (kurz: SCGI) [Sch08] verwendet.
Alternativ existieren verschiedene weitere Protokolle zum Datenaustausch zwischen WS und AS wie das auf Python-AS spezialisierte WSGI (Web Server Gateway Interface) [Eby10] oder das von Apache-Produkten verwendete Apache JServ Protocol (kurz: AJP) [The20b].

3.2 Entwurf der Enklavenarchitektur

Nachdem die A_{MS} bezüglich ihres Aufbaus detailliert untersucht wurde, folgt der Entwurf der für diese Anwendung genutzten SGX-Umgebung. Als Voraussetzung der eigentlichen Migration ist das Ergebnis dieses Schrittes die Beschreibung der A_{MS}-Architektur, welche deren Dienste und Kommunikationskanäle beinhaltet und spezifiziert, wie SGX-Enklaven genutzt werden, um die Vertraulichkeit und Integrität der kritischen Daten zu gewährleisten.

In diesem Schritt erfolgt die Festlegung der Anzahl verwendeter Enklaven sowie deren Inhalts. Dabei wird geprüft, welche Dienste jeweils in einer eigenen Secure Enclave ausgeführt und welche möglicherweise in einer gemeinsamen zusammengefasst werden. Um die TCB möglichst klein zu halten, sollte immer nur jene Anwendungsfunktionalität durch Enklaven geschützt werden, für die dies zwingend erforderlich ist. Neben der Dienstauswahl bedeutet das, alle nicht benötigten Erweiterungen bzw. Module zu deaktivieren sowie, wenn möglich, eine Partitionierung durchzuführen und nur den schützenswerte Daten verarbeitenden Teil des svc_i zu migrieren. Diese Aufgabe kann möglicherweise durch ein Partitionierungsframework (siehe Abschnitt 2.2.2) unterstützt werden.
Einhergehend mit der Aufteilung von Diensten auf Enklaven werden die in Schritt 1 (Abschnitt 3.1) bestimmten Kommunikationswege dem SGX-Szenario zugeordnet.
Im Gegensatz zum Datenaustausch zwischen mehreren Diensten in der gleichen Enklave erfordert der über das nicht vertrauenswürdige Betriebssystem verlaufende eine besondere Absicherung. Dabei unterscheidet sich die Kommunikation zwischen zwei Enklaven nur wenig von der zu einer I/O-Schnittstelle wie der Netzwerkkarte (und schließlich dem Nutzer).
Damit nur der in der Enklave ausgeführte Dienst auf die Schlüssel einer kryptografisch abgesicherten Verbindung zugreifen kann, muss diese stets innerhalb des *tCodes* terminiert und die entsprechenden Geheimnisse erst nach erfolgreicher Attestation ausgetauscht werden.

Zusammengefasst müssen im zweiten Migrationsschritt demnach folgende Designfragen beantwortet werden:

- Welche Dienste werden in einer eigenen Enklave geschützt?
- Werden mehrere Dienste in eine gemeinsame Enklave migriert?
- Ist eine Partitionierung von Diensten möglich, um nur einen Teil in der Enklave auszuführen?

- Wie kommunizieren mehrere Dienste in der gemeinsamen Enklave?
- Welche Kommunikationskanäle bestehen zwischen den verschiedenen Enklaven und zur Umwelt?
- Wie wird der Datenaustausch über das Betriebssystem abgesichert?

Beispiel: Im Beispiel einer Webanwendung sollen die drei Dienste WS, AS und DB durch SGX geschützt werden. Der Datenbankserver wird dabei in einer eigenen Enklave ausgeführt. Da Datenbanken u.U. für mehrere Anwendungen genutzt werden, soll diese DB möglichst unabhängig von den anderen Diensten sein.
Je nach eingesetzter Implementierung besteht die Möglichkeit, den Web- und Anwendungsserver entweder in eine gemeinsame oder jeden in seine eigene Enklave zu migrieren, wobei diese beiden Optionen entsprechend des spezifischen Anwendungsszenarios gegeneinander abgewogen werden müssen. Während eine gemeinsame Enklave komplexe, sichere Kommunikationsprotokolle und die gegenseitige Attestation zwischen WS und AS überflüssig macht, ermöglichen getrennte Enklaven z.B. eine stärkere Entkopplung und die Ausführung beider Anwendungsbestandteile auf unterschiedlichen physischen Rechnern.
Eine Partitionierung der drei Dienste ist aufgrund der Komplexität der jeweiligen Software nicht trivial, bei Bedarf kann aber ein entsprechendes Framework eingesetzt werden. Eine Deaktivierung aller nicht benötigten Funktionalitäten (z.B. Erweiterungen oder Module) sollte jedoch in jedem Fall erfolgen.
Die beschriebene Enklavenarchitektur beinhaltet demzufolge die gleichen Kommunikationswege wie die A_{MS}, bis auf die ggf. entfallende Kommunikation zwischen WS und AS.
Die komplette Architektur wird (beispielhaft für die Platzierung von WS und AS in einer gemeinsamen Enklave) in Abbildung 3.3 veranschaulicht.

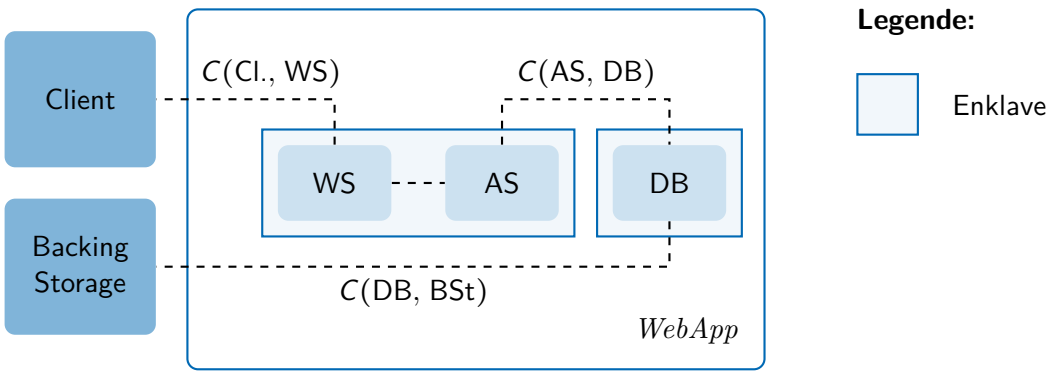


Abbildung 3.3: Enklavenarchitektur einer beispielhaften Webanwendung

3.3 Auswahl der Werkzeuge

Wurde die Zielarchitektur der Multi Service Application im SGX-Umfeld spezifiziert, folgt in einem dritten Schritt die Auswahl der zu verwendenden Migrationsansätze und Hilfsmittel.
Das Ziel ist hierbei die Fertigstellung der Migrationsvorbereitung und Bestimmung der für die jeweiligen Werkzeuge und Frameworks spezifischen Anforderungen.

Um die Auswahl der verfügbaren Ansätze zur Migration einzugrenzen, ist es empfehlenswert, sich zunächst auf eine passende Migrationsstrategie festzulegen (siehe Abschnitt 2.2.1). Anschließend werden aus den diese Strategie verfolgenden Studien bzw. Werkzeugen eine oder mehrere ausgewählt, die für den bestimmten Anwendungsfall am besten geeignet sind.
Dabei wird selten eine einzelne Entscheidung für die gesamte A_{MS} getroffen. Stattdessen sollten alle in einer Enklave auszuführenden Dienste als unabhängig voneinander betrachtet werden, so lange der in der Enklavenarchitektur zwischen ihnen geforderte Informationsaustausch gewährleistet wird. Demnach kann es u.U. sinnvoll sein, für einige Enklaven den selben, für andere jedoch einen anderen Migrationsansatz zu nutzen, wobei jeweils alle in einer Secure Enclave ausgeführten Dienste als Einheit betrachtet werden (entsprechend der Enklavenarchitektur nur einer oder auch mehrere). Da die von Multi Service Applications verwendeten Dienste jedoch sehr heterogene Eigenschaften aufweisen, kann dazu keine pauschale Aussage getroffen werden.
In Bezug auf die jeweils in einer Enklave auszuführenden Dienste sollten zur Unterstützung des Auswahlprozesses die folgenden Fragen beantwortet werden.

1. Welcher Aufwand wird als praktikabel eingeschätzt?
2. Wie groß ist die Kenntnis des Quellcodes?
3. Welche Programmiersprache wird verwendet und handelt es sich dabei um kompilierten oder interpretierten Code?
4. Wird eine Open-Source-Lösung benötigt?
5. Auf welche Weise sollen maximale Performanz und maximale Sicherheit gegeneinander abgewogen werden? – damit einhergehend:
    a) Wie wichtig ist eine Minimierung der Enklaven-TCB?
    b) Wie wichtig ist eine Minimierung der Schnittstelle zum Host-OS?
    c) Welche Strategie für Systemcalls soll verwendet werden?

6. Welche sicherheitssteigernden Maßnahmen und Funktionen werden benötigt?
7. Welche Kompatibilitätsanforderungen bestehen? – damit einhergehend:
    a) Ist der Einsatz vom Standard abweichender OS-Kernels bzw. Treiber möglich?
    b) Wird eine bestimmte C-Bibliothek für Systemcalls benötigt (z.B. *glibc* oder *musl libc*)?

Wurde die Entscheidung für einen oder mehrere Migrationsansätze getroffen, müssen außerdem deren spezifische Anforderungen bezüglich verfügbarer Informationen oder Details der Migrationsumgebung ermittelt werden. Für verschiedene Partitionierungsframeworks ist beispielsweise die Kenntnis aller Funktionen oder Variablen im Quellcode erforderlich, welche auf die schützenswerten Daten zugreifen können.

Beispiel: In Bezug auf die beispielhafte Webanwendung mit je einer Enklave für WS, AS und DB kann in diesem Zusammenhang z.B. festgestellt werden, dass alle drei Dienste relativ komplexe Fremdentwicklungen darstellen, über deren Quelltext keine weitreichenden Kenntnisse bestehen. Entsprechend müssen alle Migrationsansätze ausgeschlossen werden, welche eine Veränderung, Neuentwicklung oder Annotation des Programmcodes erfordern.
Da vor allem Datenbank-, aber auch Webserver umfangreichen Gebrauch von speziellen Systemcalls für z.B. Locking-Mechanismen machen, sollte darauf geachtet werden, dass die entsprechende C-Bibliothek sowie die Ausführungsumgebung in der Enklave alle benötigten Funktionen unterstützt. Wird die *glibc* eingesetzt, treten hierbei in der Regel keine Kompatibilitätsprobleme auf.
Da der WS bei Lastspitzen in den Nutzeranfragen mit einer hohen Frequenz auf die Netzwerkschnittstelle zugreifen muss, sollte eine Systemcall-Strategie verwendet werden, die nicht für jeden `read()`- oder `write()`-Aufruf einen Enclave Exit durchführt. Eine Partitionierung der in Enklaven ausgeführten Anwendungen ist in diesem Szenario nicht ohne Weiteres möglich, da alle drei Dienste in sich monolithisch aufgebaut sind und die Mehrheit ihrer funktionalen Teile Zugriff auf schützenswerte Daten hat (z.B. beim DB-Server). Die Funktionalität (und damit die Möglichkeit zum Zugriff auf solche Daten) des Anwendungsservers wird erst zur Laufzeit durch das ausgeführte Programm (z.B. ein PHP-Skript) festgelegt, weshalb vorab keine Teile des AS als irrelevant aus der Enklave exkludiert werden können.

Aufgrund dieser Betrachtungen kommen z.B. die Shielding Layers „SCONE“ und „ScriptShield“ in Betracht. Dabei kann Erstere nur verwendet werden, wenn die Funktionalität der *musl libc* ausreicht und Letztere, wenn der benötigte Interpreter

bereits portiert wurde. Weiterhin ist der Einsatz diverser Implementierungen eines Library OS möglich. Insbesondere kann eine Entscheidung zwischen den Alternativen „Graphene-SGX“ (bei benötigter *glibc*-Bibliothek), „SGXKernel“ (mit asynchronen Systemcalls) und „Occlum“ (zur Unterstützung von Multitasking-Szenarien) erfolgen. Die beiden anderen libOS-Ansätze wurden dagegen durch oben genannte Kriterien ausgeschlossen.
Anschließend werden die Besonderheiten und Anforderungen jedes gewählten Migrationswerkzeugs bestimmt. So benötigt z.B. Graphene-SGX zum Ausführen einer Anwendung eine sog. *Manifest*-Datei, in der alle in die Enklave zu ladenden Dateien und Bibliotheken anzugeben sind.

3.4 Implementierung

In diesem letzten Schritt erfolgt schließlich die praktische Umsetzung der vorbereiteten und geplanten Migration.
Das Ziel ist hierbei die Ausführung der A_{MS} im Umfeld von SGX-Enklaven nach der in Schritt 2 (Abschnitt 3.2) festgelegten Architektur, wobei alle funktionalen Eigenschaften der Anwendung erhalten bleiben sollen, die Integrität und Vertraulichkeit der – in Schritt 1 (Abschnitt 3.1) definierten – schützenswerten Daten durch Intel SGX aber gesichert wird.

Die praktische Durchführung der gesamten Migration sollte sequenziell in mehreren Teilschritten (je einer für jeden in einer Enklave auszuführenden Dienst) erfolgen, da möglicherweise verschiedene Migrationsansätze zum Einsatz kommen und so die Komplexität reduziert werden kann. Außerdem kann es hilfreich sein, nach jeder dieser Teilmigrationen einen Funktionstest der gesamten Multi Service Application durchzuführen, um eventuelle Fehler schnellstmöglich detektieren und beheben zu können.
Von besonderer Bedeutung bei der Implementierung der spezifizierten Enklavenarchitektur ist die Unterstützung der SGX-Funktionalität zur Attestierung von Secure Enclaves (siehe Abschnitt 2.1.6.1). Dabei ist eine Remote Attestation zu bevorzugen, um die Integrität der Enklavenumgebung über das Netzwerk überprüfen zu können und die Flexibilität der Ausführung einzelner Dienste auf unterschiedlichen physischen Rechnern zu gewährleisten.

4 Beispielhafte Migration von WordPress

Nach der allgemeinen Beschreibung der Methodik zur Migration einer Multi Service Application in das Umfeld von Intel SGX wird in diesem Kapitel deren Implementierung und Evaluation am Beispiel einer spezifischen Webanwendung beschrieben.
Dazu kommt das als Open Source vorliegende Content-Management-System (kurz: CMS) „WordPress“ [Aut20] zum Einsatz, welches z.Zt. von ca. 38% der meistbesuchten Seiten im Internet zur Darstellung ihrer Inhalte eingesetzt wird und einen Marktanteil von rund 63% aller bekannten CMS aufweist [QSu20].
Abhängig von den Möglichkeiten und Anforderungen des jeweiligen Einsatzszenarios kann WordPress auf sehr diverse Weise installiert und verwendet werden, wobei an dieser Stelle von der Basis-Installation entsprechend der offiziellen Anleitung [Wor20c] ausgegangen wird.

4.1 Durchführung der Migration

Das Ziel der Migration von WordPress ist der Schutz anwendungsbezogener Daten vor Angriffen auf die Vertraulichkeit und Integrität durch einen privilegierten Angreifer oder Innentäter.
In diesem Abschnitt soll das detaillierte Vorgehen entsprechend der in Kapitel 3 vorgestellten Methodik geschildert werden.

4.1.1 Analyse der Anwendungsstruktur

Kritische Daten Während WordPress ursprünglich als Werkzeug zum einfachen Erstellen von Blogs oder statischen Webseiten entwickelt wurde [McK13, S. 1 ff.], kann es durch den Einsatz zahlreicher Plugins um viele verschiedene Funktionen (u.a. die eines vollwertigen CMS, Forums oder sozialen Netzwerks) erweitert werden. Demzufolge sind die zu schützenden Daten stark abhängig vom spezifischen Einsatzzweck und den aktivierten Plugins.

In diesem Beispiel wird davon ausgegangen, dass die Software ausschließlich zum Verbreiten von Blogbeiträgen (sog. „Posts“) mit öffentlicher Kommentarfunktion

verwendet wird, es werden also keine zusätzlichen Plugins installiert.
Da ein typischer Post nach seiner Erstellung öffentlich einsehbar ist, besteht für die enthaltenen Informationen keine Notwendigkeit des Vertraulichkeitsschutzes. WordPress bietet allerdings auch die Möglichkeit, private Beiträge zu erstellen, die nur einem begrenzten Publikum zugänglich gemacht werden [Wor20b]. Solche Posts sowie Konfigurationsinhalte wie Zugangsdaten zur zugrundeliegenden Datenbank bedürfen des Schutzes vor Angriffen auf die Vertraulichkeit. Eine Gewährleistung der Datenintegrität ist dabei in jedem Fall erforderlich – so sollen weder vertrauliche Konfigurationsdaten noch öffentlich (oder nur nach Einladung) einsehbare Inhalte von unautorisierten Nutzern geändert werden können.
Wird das CMS z.B. in einer Public-Cloud-Umgebung installiert, bedeutet dies, dass nur der Webseitenadministrator Blogbeiträge erstellen, aktualisieren oder löschen sowie die Einstellungen der WordPress-Umgebung ändern darf. Ein Administrator des physischen Rechners oder des Betriebssystems soll auf diese Daten keinen Zugriff erhalten.

Dienste Die Anwendungsfunktionalität von WordPress wird als Projekt von PHP-Skriptdateien bereitgestellt (siehe [Doc20c] und [Wor20f]).
Zu deren Ausführung muss ein Interpreter dieser Skriptsprache zum Einsatz kommen, welcher Anwendungsserver arbeitet. Um WordPress als vollständige WebApp nutzen zu können, werden zusätzlich die beiden anderen Bestandteile der in Abschnitt 3.1 vorgestellten „3-Tiered-Architecture“ benötigt. Ein Webserver realisiert dabei die Interaktion mit dem Nutzer und durch Verwendung einer Datenbank wird eine persistente Datenspeicherung ermöglicht. Folglich kann die Webanwendung WordPress in ihrer Gesamtheit als Multi Service Application betrachtet werden.
In einigen Fällen unterstützt das CMS allerdings nur bestimmte Dienstimplementierungen (sowie deren Versionen), was zu dem folgenden A_{MS}-Aufbau führt [Doc20c; Wor20f]:

Webserver:	beliebige Implementierung bei Unterstützung einer korrekten Anbindung des AS
Anwendungsserver:	PHP ab Version 7.3
Datenbankserver:	MySQL ab Version 5.6 bzw. MariaDB ab Version 10.1

Dienstimplementierungen Da WordPress dem Nutzer freie Wahl in Bezug auf den verwendeten Webserver lässt, wird im Folgenden der für seine geringen Speicher- und CPU-Anforderungen bekannte WS „Lighttpd“ [Lan20] eingesetzt.
Um den von WordPress auch bei geringer Nutzerzahl teilweise parallel gestellten

Anfragen an den AS[1] flexibel begegnen zu können, wurde die Implementierung „PHP-FPM“ (FastCGI Process Manager) gewählt. Sie stellt einen (statisch oder dynamisch) skalierten Workerpool mit PHP-Prozessen bereit, welche Anfragen über FastCGI entgegennehmen, und ist so für die Bearbeitung vieler Anfragen in kurzer Zeit geeignet [The20e].
Als Datenbank kommt in diesem Beispiel die Open-Source-Variante MariaDB zum Einsatz.

Relevante Dienste Der Zugriff auf kritische Daten ist von allen drei Diensten möglich, weshalb sowohl der Web- und Anwendungs- als auch der Datenbankserver durch Enklaven geschützt werden müssen.
Ersterer erlangt Kenntnis von allen Nutzeranfragen sowie den vom AS ausgegebenen Antworten. Der PHP-Interpreter hat Zugriff auf alle Ein- und Ausgabedaten der Anwendung und zum Organisieren seiner Inhalte benötigt der DB-Server ebenfalls alle Daten in unverschlüsselter Form.
Somit ist eine Migration aller drei Dienste erforderlich.

Dienstabhängigkeiten Nach der Identifikation durch Enklaven zu schützender svc_i folgt die Ermittlung deren Abhängigkeiten, Programmdateien und Bibliotheken. Dazu ist es notwendig, jeden der drei WordPress-Dienste entsprechend der Anforderungen dieser Webanwendung zu kompilieren und zu konfigurieren. Auf die jeweiligen Einzelheiten wird in Abschnitt 4.1.4 genauer eingegangen.
Die Bestimmung aller Abhängigkeiten eines A_{MS}-Dienstes kann auf Grundlage der nachfolgend erläuterten Schritte vorgenommen werden.

Programmdatei Die zum Starten des Dienstes verwendete ausführbare Datei kann in der Regel mit Hilfe der Dokumentation oder Installationsanleitung detektiert werden. Ist der Dienst auf einem Linux-Rechner installiert, ermittelt der folgende Befehl deren absoluten Speicherort.

```
which <Dateiname>
```

Dynamische Bibliotheken Benötigt ein Dienst zu seiner Ausführung extern bereitgestellte Funktionen, erfolgt die Verknüpfung mit den entsprechenden Inhalten von Programmierbibliotheken während des auf die Programmkompilierung fol-

[1] Neben den eigentlichen Nutzeranfragen an die Webanwendung führt WordPress wiederkehrende Verwaltungsaufgaben (z.B. Prüfung auf Updates) automatisiert aus und erzeugt somit zusätzliche Anfragen an den AS. In Tests hat sich gezeigt, dass dadurch u.U. beim Aufruf einer WordPress-Seite mehrere parallele PHP-Verarbeitungen angefordert werden, die von einem einzelnen PHP-Prozess nicht schnell genug bearbeitet werden können.

genden Linking-Prozesses.

Solche Funktionen können entweder (im Falle statischer Bibliotheken) direkt in die ausführbare Datei eingefügt oder (bei *shared libraries*) erst zur Laufzeit durch den sog. „Loader“ in den Speicher geladen werden [Tan09, S. 277 ff.]. Nur in letzterem Fall müssen die verwendeten Bibliotheken vor der Ausführung in einer SGX-Enklave bekannt sein, da alle statisch gelinkten Funktionen bereits in der Anwendung enthalten sind und gemeinsam mit dieser geladen werden.

Auf einem Linux-System[2] können alle zum Starten einer Programmdatei erforderlichen dynamischen Bibliotheken mit dem folgenden Befehl aufgelistet werden.

```
ldd <Programmdatei>
```

Weitere Bibliotheken Alle drei verwendeten Dienste ermöglichen das Hinzufügen zusätzlicher Funktionalität durch weitere (bei Lighttpd, PHP und MariaDB jeweils unterschiedlich als „Module“, „Extensions“ resp. „Plugins“ bezeichnete) Bibliotheken. Diese werden von der jeweiligen Anwendung selbst, erst nach Programmstart, geladen und deshalb nicht durch den `ldd`-Befehl erkannt.

Entsprechend müssen solche Bibliotheken manuell ermittelt werden. Einen wichtigen Anhaltspunkt für den Einsatz von Modulen bzw. Erweiterungen stellen die Konfigurationsdateien des jeweiligen Dienstes dar. Außerdem können einige dieser Zusatzbibliotheken wiederum weitere dynamische Abhängigkeiten aufweisen, weshalb auch auf diese das `ldd`-Kommando angewendet werden sollte. Abschließend kann die Auflistung benötigter *shared libraries* jedoch oft nur durch Versuch und Irrtum[3] vervollständigt werden.

Dateien Auch alle weiteren Dateien, die ein Dienst möglicherweise in seine Enklave lädt, müssen zur Integritätsprüfung vor deren Start bekannt sein. Zu diesen Dateien zählen insbesondere Konfigurationsdateien, welche zum Start der jeweiligen Anwendung benötigt werden und die ihrerseits oft Hinweise auf weitere verwendete Verzeichnisse bzw. Dokumente (z.B. Logdateien oder Anwendungsdaten) enthalten. So benötigen beispielsweise sowohl Web- als auch Anwendungsserver

[2] Auf dem eingesetzten Testsystem mit 64-Bit-Architektur wird die Linux-Distribution „Ubuntu“ in der Version 18.04 LTS verwendet.

[3] Um die zur Laufzeit benötigten Bibliotheken (oder andere Dateien) einer Anwendung zu ermitteln, kann z.B. eine Migration mit allen bereits bekannten Abhängigkeiten erfolgen. Anschließend wird versucht, den entsprechenden Dienst in einer Enklave zu starten.
Wurden noch nicht alle erforderlichen Dateien angegeben, bricht die Ausführung in der Regel mit einer auf die fehlende(n) Datei(en) verweisenden Fehlermeldung ab und die Abhängigkeitsliste kann somit erweitert werden.
Zusätzlich können alle von dem jeweils verwendeten Migrationsansatz bereitgestellten Debugging-Funktionen genutzt werden, um weitere Informationen zu benötigten Dateien und Verzeichnissen zu erhalten.

Zugriff auf die Applikationsdateien der WebApp.
Auch eine vollständige Auflistung benötigter Entitäten im Dateisystem erfordert oft manuelle Funktionstests und die Auswertung von Fehlermeldungen.

Die für jeden der drei WordPress-Dienste ermittelten Abhängigkeiten werden in Abschnitt B.1, Anhang B strukturiert aufgelistet.

Schnittstellen Die in Abschnitt 3.1 beschriebenen Kommunikationskanäle gelten auch für die Webanwendung WordPress. In diesem Fall wird der Datenaustausch zwischen WS und AS als C(Lighttpd, PHP-FPM) und jener zwischen AS und DB als C(PHP-FPM, MariaDB) bezeichnet. Zusätzlich kommuniziert der Webserver mit den Nutzern der Anwendung und die MariaDB greift auf ihren Backing Storage – ein Verzeichnis mit Binärdateien auf einem entsprechenden Datenträger – zu. Erstere Kommunikation wird als C(Client, Lighttpd) und letztere als C(MariaDB, Datenbasis) bezeichnet.
Nachfolgend werden die Besonderheiten dieser Schnittstellen sowie verfügbare Protokolle und Sicherheitsmaßnahmen thematisiert.

C(*Client*, *Lighttpd*) Vom WordPress-Nutzer an die Webanwendung gestellte Anfragen werden als Erstes vom Webserver Lighttpd verarbeitet, der für diese Kommunikation in der Regel TCP-Sockets verwendet.
Der eigentliche Datenaustausch erfolgt dabei mit Hilfe von Dateien im HTML-Format, die nach dem HTTP-Protokoll übermittelt werden, wobei optional eine Verschlüsselung durch den Einsatz von TLS erreicht werden kann (als HTTPS bezeichnet). Eine so verschlüsselte Verbindung wird vom WS terminiert und muss in dessen Konfiguration aktiviert werden.
Auch die verwendeten Ports können nach Bedarf eingestellt werden, wobei standardmäßig der TCP-Port 80 (für HTTP) bzw. 443 (für HTTPS) zum Einsatz kommt.

C(*Lighttpd*, *PHP-FPM*) Der Datenaustausch zwischen dem Web- und Anwendungsserver kann auf verschiedene Weise erfolgen. Da für Lighttpd kein spezielles PHP-Modul zur Verfügung steht[4], muss der AS stattdessen unter Verwendung einer Variante des CGI-Protokolls als eigenständiger Server ausgeführt werden.
Das klassische Common Gateway Interface ist für den Einsatz im SGX-Umfeld jedoch

[4]Für die Nutzung von PHP in Kombination mit dem Apache-Webserver „httpd“ [The20a] existiert beispielsweise ein Modul zur Integration des PHP-Interpreters als dynamische Bibliothek in den WS. Da der AS dadurch in jedem httpd-Prozess enthalten ist, entfällt der Aufwand zum Starten des Interpreters, dafür ist die Funktionalität jedoch in allen WS-Threads vorhanden, selbst wenn sie in einzelnen Anfragen nicht benötigt wird [The19].

nicht geeignet, da es einen PHP-Prozess jeweils nur für eine einzelne Anfrage des WS (z.B. Ausführung einer Skriptdatei) verwendet. Für jede weitere Berechnung muss demnach ein neuer Prozess – und damit eine neue Enklave – gestartet werden, was aufgrund des SGX-Verwaltungsaufwandes sehr starke Performanzeinbußen mit sich bringt.
Bei der Verwendung von FastCGI wird der PHP-Interpreter dagegen nur einmal gestartet, der Prozess bleibt anschließend aktiv und kann zur Bearbeitung weiterer Anfragen verwendet werden, weshalb diese Alternative zu bevorzugen ist.
Ein PHP-Prozess im FastCGI-Modus, der über Unix- oder TCP-Sockets entsprechende Anfragen entgegennimmt, kann entweder durch die Programmdatei `php-cgi` manuell oder durch den PHP FastCGI Process Manager automatisiert gestartet werden. Letzterer verwendet einen Workerpool aus einem oder mehreren PHP-Prozessen und ermöglicht dessen automatische und bedarfsgerechte Skalierung. Deshalb empfiehlt sich diese Alternative besonders für Anwendungen mit Parallelitätsanforderungen und stark wechselnden Auslastungsniveaus.
Eine Möglichkeit zur Verschlüsselung der übertragenen Daten ist im FastCGI-Protokoll jedoch nicht vorgesehen, weshalb alle Ein- und Ausgabedaten der PHP-Verarbeitung im Klartext übermittelt werden.

C(*PHP-FPM*, *MariaDB*) Werden für die Webanwendung Daten aus der durch MariaDB verwalteten Datenbank (z.B. Informationen über WordPress-Nutzer oder Inhalte von Posts) benötigt, wird durch den AS – je nach Konfiguration unter Linux entweder über einen Unix- oder TCP-Socket[5] – eine Verbindung zum Datenbankserver aufgebaut.
Die Kommunikation zwischen der DB und dem AS erfolgt dabei über das dienstspezifische MariaDB- bzw. MySQL-Protokoll (Details unter [Mar20a]), welches in der Grundeinstellung alle Inhalte der Clientanfragen und Serverantworten unverschlüsselt übermittelt. Nach einer Aktivierung in der Konfigurationsdatei des DB-Servers und der Bereitstellung der benötigten Zertifikate kann diese Kommunikation jedoch ebenfalls mittels TLS-Verschlüsselung abgesichert werden (siehe [Mar20d]).

C(*MariaDB*, *Datenbasis*) Für den Lese- und Schreibzugriff auf seine Datenbasis (ein Verzeichnis mit diversen Binär- und Textdateien) verwendet MariaDB Standard-Systemcalls, die vom Kernel ausgeführt werden. Demnach existiert kein spezifisches „Kommunikationsprotokoll“ abseits der konventionellen Syscall-Behandlung – und somit keine Möglichkeit zur Verschlüsselung.

[5] Unter Windows kann die Kommunikation mit dem MariaDB-Server dagegen über einen TCP-Socket, gemeinsamen Speicher oder eine benannte Pipe erfolgen [Mar20b].

Zusammenfassung Um die Erkenntnisse dieses Migrationsschrittes zusammenzufassen, wird der Aufbau von WordPress als beispielhafte A_{MS} in Abbildung 4.1 dargestellt.

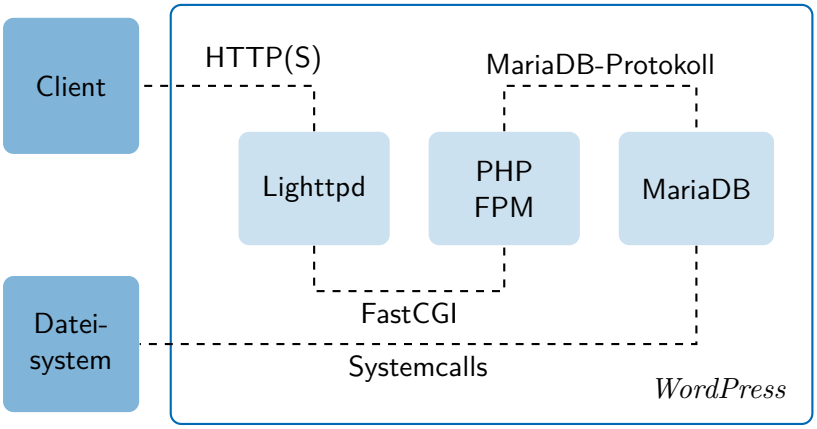


Abbildung 4.1: Aufbau der Multi Service Application WordPress

4.1.2 Entwurf der Enklavenarchitektur

Da in Abschnitt 4.1.1 festgestellt wurde, dass alle drei Dienste der A_{MS} WordPress des Schutzes durch SGX-Enklaven bedürfen, müssen sowohl WS und AS als auch die DB in der Enklavenarchitektur enthalten sein.
Dabei wird jeder Dienst in eine eigene Enklave migriert, weil der PHP-Interpreter nicht als Bibliothek in den Webserver Lighttpd integriert werden kann und um die verteilte Ausführung der verschiedenen $A_{MS,p}$-Komponenten auf differenten Rechnern zu unterstützen. Weiterhin soll jeder vom FPM gestartete PHP-Prozess durch eine eigene Enklave geschützt werden, um eine Separierung verschiedener Anfragen – und damit auch verschiedener Nutzer – zu erreichen.
Aufgrund fehlender Kenntnisse über die Quellcode-Interna der drei komplexen Dienste sowie der im Beispiel für Migrationsschritt 3 (Abschnitt 3.3) aufgeführten Gründe wird eine Partitionierung als zu aufwendig angesehen. Dennoch müssen die jeweils aktivierten Module, Erweiterungen bzw. Plugins auf das zwingend erforderliche (in der WordPress-Dokumentation unter [Wor20e] angegebene) Minimum reduziert werden.

Da aufgrund dieser Enklavenaufteilung jegliche zuvor bestimmten Kommunikationskanäle über das nicht vertrauenswürdige Betriebssystem verlaufen, müssen sie alle in der Enklavenarchitektur berücksichtigt und abgesichert werden.
Dabei sollten die vorhandenen Sicherheitsmaßnahmen (HTTPS und TLS-verschlüsseltes MariaDB-Protokoll) Anwendung finden. Für die Kommunikation C(Lighttpd, PHP-FPM) über das FastCGI-Protokoll besteht jedoch keine Möglichkeit zur Ver-

schlüsselung, sodass eine alternative Absicherung benötigt wird. Möglicherweise kann diese durch die eingesetzten Migrationswerkzeuge erreicht werden.
Der Zugriff auf die Inhalte der Datenbasis darf ausschließlich für die in der Enklave ausgeführte Datenbank erlaubt sein. Um dies zu erreichen, ist die konsequente Verschlüsselung der entsprechenden Dateien sowohl in persistierter Form auf dem Datenträger als auch während der Übertragung durch den Kernel vom Dateisystem an die DB-Anwendung erforderlich. Da diese eine solche Verschlüsselung nicht (ausreichend) unterstützt, wird auch dabei externe Unterstützung benötigt.

Die Architektur der migrierten WordPress-Anwendung wird in Abbildung 4.2 zusammengefasst.

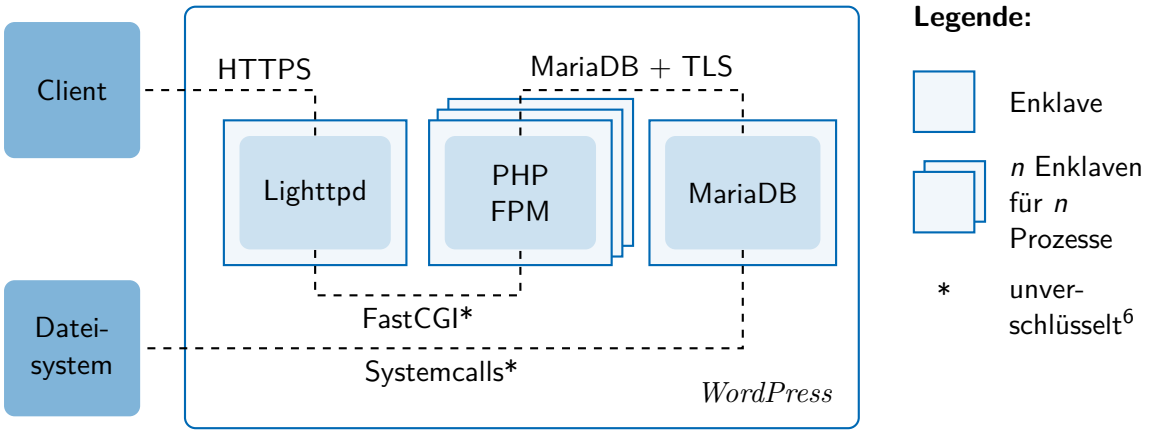


Abbildung 4.2: Enklavenarchitektur von WordPress

4.1.3 Auswahl der Werkzeuge

Bei den drei für WordPress benötigten Diensten handelt es sich um komplexe Fremdentwicklungen, weshalb für deren Migration der Aufwand einer Quellcodemodifikation, -annotation oder Neuentwicklung als zu hoch eingeschätzt wird. Da die Enklavenarchitektur außerdem keine Partitionierung vorsieht, können alle Migrationsstrategien ausgeschlossen werden, welche Programmierbibliotheken oder Automatisierungsframeworks verwenden.
Das bloße Rekompilieren wird dagegen als praktikabel angesehen, weil sowohl WS und AS als auch die DB als Open-Source-Software zur Verfügung stehen. Außerdem wird in diesem Testszenario der Kompatibilität des verwendeten Migrationsansatzes mit den Anforderungen der portierten Dienste mehr Priorität beigemessen als einer Minimierung der Enklaven-TCB. Folglich stellt sowohl die Shielding Layer als auch

[6]Bei diesen Kommunikationsverbindungen werden weiterführende Schutzmaßnahmen benötigt, da das jeweilige Protokoll keine Verschlüsselung unterstützt.

das Library OS eine valide Strategie zur Migration von WordPress dar.

Wahl der Migrationsansätze Bei der Auswahl geeigneter Shielding Layers wurde festgestellt, dass das auf die Ausführung von Interpreter-basierten Anwendungen spezialisierte „ScriptShield“ theoretisch für die Migration des Anwendungsservers genutzt werden könnte. Tatsächlich wird die Sprache PHP von diesem (zum Zeitpunkt der Erstellung dieser Arbeit) allerdings nicht unterstützt.
Weiterhin zeigt die „SCONE“-Dokumentation [SCO20d] auf, dass dessen Entwickler bereits eine in SGX migrierte Version von MariaDB als Docker-Container bereitstellen. Die Migration des Datenbankservers kann also durch die Verwendung dieses Containers abgekürzt werden, dessen Software zwar nicht als Open Source, aber in der „Community-Version“ kostenfrei zur Verfügung steht.

Zur Portierung des Web- und Anwendungsservers soll dagegen ein libOS zum Einsatz kommen, welches (1.) auf einem Linux-System ausgeführt werden kann, (2.) als Open-Source-Software bereitgestellt wird sowie (3.) in der Praxis erprobt ist und akzeptable Einbußen bezüglich der Performanz mit sich bringt.
Die ersten beiden Kriterien werden von drei Library-OS-Implementierungen erfüllt; „SGX-LKL“, „Graphene-SGX“ und „Occlum“. Ersteres beeinflusst die Ausführungsgeschwindigkeit des *tCodes* aufgrund der Priorisierung einer minimalen Schnittstelle zum Betriebssystem jedoch so stark, dass ein Einsatz in der Praxis nicht möglich ist. Von den verbliebenen zwei Alternativen wird schließlich Graphene-SGX (eine auf das Enklavenumfeld angepasste Variante des libOS „Graphene“) gewählt, da es als Open-Source-Projekt die größte Verbreitung und aktivste Entwicklung[7] aufweist und vielfach als „De-facto-Standard“ eines Library OS eingesetzt wird (Stand: 11/2020). Das relativ junge Occlum-Projekt könnte allerdings zukünftig an Bedeutung gewinnen und eine äquivalente Alternative darstellen.

Spezifika und Anforderungen Bei der praktischen Durchführung der Migration jedes einzelnen WordPress-Dienstes müssen die jeweiligen Anforderungen der spezifischen Ausführungsumgebungen (in diesem Fall SCONE und Graphene-SGX) berücksichtigt werden.

SCONE Zur Migration von kompilierten Anwendungen mittels SCONE wird in der Regel ein Docker-Container benötigt, in dem eine Rekompilierung der entsprechenden Quelltexte unter Nutzung eines speziellen (u.a. gegen die *musl*-Bibliothek linkenden) SCONE-Compilers durchgeführt wird. Anschließend kann eine solche Anwendung

[7] Im Zeitraum vom 13.10.2019 bis 03.10.2020 wurden jede Woche durchschnittlich 16 Commits zu dem GitHub-Repository von Graphene hinzugefügt [Git20b].

durch das Setzen bestimmter Umgebungsvariablen[8] von der SCONE-Laufzeitumgebung in einer SGX-Enklave ausgeführt werden [SCO20a].
Da der Datenbankserver MariaDB jedoch bereits als SCONE-Variante zur Verfügung steht, entfallen hierbei die Schritte zur Übersetzung des Quellcodes. Stattdessen kann der entsprechende Container (eine korrekte Docker-Installation vorausgesetzt) direkt zur Ausführung gebracht werden.
In jedem Fall muss die Schnittstelle zum SGX-Treiber des Hostsystems innerhalb des Containers verfügbar sein, je nach eingesetztem Attestation-Schema (IAS oder DCAP) werden dabei andere Blockdevices unter `/dev/` eingebunden. Außerdem benötigt der DB-Server im Container verschiedene Informationen (z.B. ein Passwort für den standardmäßig erstellten Nutzer *root*), welche im einfachsten Fall mittels Umgebungsvariablen, im Produktiveinsatz allerdings nur durch Secret Provisioning übermittelt werden.

Die Kommunikation zwischen der Datenbank und deren Backing Storage kann durch die Verwendung der SCONE-Funktionalität *File System Shield* abgesichert werden. Dabei realisiert die Shielding Layer eine für MariaDB transparente Verschlüsselung von im Voraus spezifizierten Dateien und Verzeichnissen, wobei eine Entschlüsselung nur innerhalb der Enklave möglich ist.
Um diese Funktion zu nutzen, ist die Erstellung zweier SCONE-spezifischer Dateien mit Informationen zu dem verschlüsselten Bereich des Dateisystems sowie weiteren Metadaten erforderlich. Erstere Datei wird dabei als „File System Protection File“ (kurz FSPF) bezeichnet.
Da das *File System Shield* nicht auf leere Verzeichnisse angewendet werden kann, muss für dessen Verwendung zuerst die Datenbasis initialisiert und anschließend verschlüsselt werden. Erst danach ist das Starten des eigentlichen MariaDB-Containers möglich, wobei das FSPF in diesem verfügbar gemacht und die Funktion in der SCONE-Laufzeitumgebung mit Hilfe von Umgebungsvariablen aktiviert werden muss [SCO20f].
Im produktiven Einsatz ist außerdem ein Schutz der TLS-Zertifikate und -Schlüssel auf ähnliche Weise erforderlich, außerdem sollte statt der Verwendung von Umgebungsvariablen eine SGX-Attestation mit anschließendem Secret Provisioning zur Übermittlung der verschiedenen Schlüssel durchgeführt werden [SCO20g].

[8] Diese Shielding Layer interpretiert den Wert verschiedener, zur dessen Konfiguration genutzter, Umgebungsvariablen, z.B. `SCONE_MODE`, `SCONE_ALPINE` oder `SCONE_FORK` [SCO20e].

Graphene-SGX Auch das Library OS benötigt Zugriff auf den SGX-Treiber im `/dev/`-Verzeichnis. Weiterhin ist allerdings die Verwendung eines Linux-Kernels mit speziellen Patches[9] oder alternativ die zusätzliche Installation eines Graphene-spezifischen Treibers erforderlich.
Während die eigentliche Anwendung zur Migration in eine Enklave mittels Graphene-SGX nicht modifiziert werden muss, benötigt das libOS diverse Informationen zu dessen Ausführung in einer *Manifest*-Datei. Diese beinhaltet Wertzuweisungen zu verschiedenen Konfigurationsvariablen, welche in der Dokumentation genauer beschrieben werden [Gra19d].
Im Fall von Lighttpd und PHP-FPM muss das *Manifest* mindestens die folgenden Variablen enthalten.

`loader.exec`	zu startende Programmdatei
`fs.mount.*`	Dateien oder Verzeichnisse des Hosts, die innerhalb des libOS zugänglich sein sollen
`sgx.enclave_size`	allozierter EPC-Speicher (nur für SGX-Version 1)
`sgx.thread_num`	maximale Thread-Anzahl in der Enklave (nur für SGX-Version 1)
`sgx.trusted_files.*`	in der Enklave benötigte und auf Integrität geprüfte Dateien (z.B. dynamische Bibliotheken)
`sgx.allowed_files.*`	in der Enklave benötigte, veränderbare Dateien oder Verzeichnisse (ohne Integritätsprüfung)
`sgx.protected_files.*`	in der Enklave benötigte Dateien oder Verzeichnisse, die mittels Sealings verschlüsselt werden

Vor der eigentlichen Ausführung einer Anwendung in Graphene-SGX werden (unter Nutzung der Hilfsprogramme `pal-sgx-sign` und `pal-sgx-get-token`) weitere Dateien mit für die Enklave benötigten Metadaten[10] erstellt. Die dazu verwendeten Informationen kommen aus dem angefertigten *Manifest*.

[9] Graphene-SGX benötigt die Kernelfunktionalität „FSGSBASE“, die im offiziellen Linux-Kernel nicht enthalten ist, nach Angaben der Entwickler aber voraussichtlich in der Zukunft hinzugefügt wird [Gra19a].

[10] So benötigt Graphene-SGX zum Erstellen der Zielenklave deren kryptografische Signatur und eine Datei für den Enklaventoken sowie die Hashwerte aller `trusted_files` für deren Integritätscheck [TPV17, S. 648].

4.1.4 Implementierung

Nachdem die Vorbereitung an dieser Stelle abgeschlossen wurde, kann die praktische Umsetzung der Migration von WordPress mit den beschriebenen Werkzeugen erfolgen. In den folgenden Abschnitten werden die notwendigen Schritte und Kommandos aufgezeigt, um alle drei Dienste sequenziell in SGX-Enklaven zu migrieren sowie die beschriebenen Kommunikationskanäle abzusichern.
Die Anwendungsdaten von WordPress wurden dazu heruntergeladen und in einem von allen drei Diensten zugreifbaren Verzeichnis (im Folgenden als `<WPDir>` bezeichnet) abgespeichert.

4.1.4.1 MariaDB

Migration Da für den Datenbankserver bereits ein einsatzbereites Container-Image verfügbar ist, muss dieses auf dem Zielrechner lediglich heruntergeladen, konfiguriert und ausgeführt werden.
Das Passwort für den *root*-Benutzer der Datenbank (`<root_PW>`) wird hierbei als Umgebungsvariable beim Start des Containers angegeben, außerdem ist zur Persistenz der Datenbasis über Containerneustarts hinweg das Einbinden eines entsprechenden Hostverzeichnisses (in diesem Fall der Ordner `database/`) erforderlich. Weiterhin kann bereits beim Start des Containers ein WordPress-Nutzer und eine -Datenbank erstellt werden. Zur Übermittlung der entsprechenden Informationen kommen ebenfalls Umgebungsvariablen zum Einsatz. Soll der MariaDB-Server auch von anderen Rechnern erreichbar sein, wird zusätzlich die Angabe einer Portzuweisung (und damit die automatische Erstellung entsprechender Firewallregeln durch den Docker-Dienst) benötigt.
Um die Anwendung durch den Befehl `docker run` (aus Listing 4.2) im Container zu starten, muss der SGX-Treiber schließlich in diesem – je nach Attestation-Schema mit einer der folgenden Angaben – verfügbar gemacht werden.

IAS: `--device=/dev/isgx`

DCAP: `-v /dev/sgx:/dev/sgx`

Absicherung der Kommunikation Im Zusammenhang mit diesem WordPress-Dienst muss die Kommunikation C(PHP-FPM, MariaDB) sowie C(MariaDB, Datenbasis) kryptografisch abgesichert werden.

Das datenbankspezifische Protokoll für Anfragen vom und Antworten an den Anwen-

dungsserver unterstützt zu diesem Zweck die Verschlüsselung mittels TLS.
Entsprechend der unter [Mar20d] verfügbaren Anleitung werden dazu mit Hilfe der in Listing B.1 (Anhang B) aufgeführten Kommandos zuerst die notwendigen, selbstsignierten X.509-Zertifikate generiert[11] und im Ordner `db-certs/` gespeichert. Anschließend können die Dateipfade dieser Zertifikate in der MariaDB-Konfigurationsdatei `mariadb-server.cnf` eingetragen werden, deren Inhalt in Listing B.2 dargestellt wird.

Zur Absicherung der Datenbasis und der Kommunikation C(MariaDB, Datenbasis) wird das SCONE *File System Shield* verwendet.
Dafür muss zuerst das Verzeichnis des Backing Storage sowie die für WordPress benötigte Datenbank initialisiert werden.
Zu diesem Zweck wird der MariaDB-Container mit dem in Listing 4.1 aufgeführten Kommando gestartet und die zur Unterstützung von TLS erstellte Konfigurationsdatei sowie die Verzeichnisse `database/` und `db-certs/` in den Container eingebunden. Anschließend wird dieser jedoch gleich wieder beendet, da vor der eigentlichen Ausführung des DB-Servers das SCONE *File System Shield* vorbereitet werden muss.

```
1  mkdir database  # leeres Verzeichnis für Datenbasis erstellen
2  docker run -d --rm \
3    --name mariadb-init \
4    # erstelltes Verzeichnis einbinden
5    -v $PWD/database:/var/lib/mysql \
6    # Konfigurationsdatei und Zertifikate einbinden
7    -v $PWD/mariadb-server.cnf:/etc/my.cnf.d/mariadb-server.cnf↩
      ↪ \
8    -v $PWD/db-certs:/etc/mysql-ssl \
9    # root-Passwort für die DB festlegen
10   -e MYSQL_ROOT_PASSWORD="<root_PW>" \
11   # WordPress-DB und -Nutzer anlegen
12   -e MYSQL_DATABASE="wordpress" \
13   -e MYSQL_USER="wp" \
14   -e MYSQL_PASSWORD="wordpress" \
15     # SCONE-Image für MariaDB
16     "sconecuratedimages/apps:mariadb-alpine"
17 # Beenden des Containers (nach Abschluss der Initialisierung)
18 docker stop mariadb-init
```

Listing 4.1: Initialisierung der MariaDB-Datenbank

[11] Dabei wird je ein Zertifikat für eine eigene Certificate Authority (kurz: CA) sowie den DB-Server selbst generiert. Letzterer akzeptiert diese allerdings nur, wenn sich der Zertifikatseintrag „Common Name“ (kurz: CN) von dem der CA unterscheidet [Git20a].

Wurde die Initialisierung abgeschlossen, folgt die Erstellung des SCONE-FSPF und Verschlüsselung der Datenbasis. Dazu werden die Kommandos aus Listing B.3 (Anhang B) verwendet, durch welche die Dateien `db.pb` und `keytag` im aktuellen Verzeichnis gespeichert werden.

Ausführen des DB-Servers in SCONE Nachdem das *File System Shield* konfiguriert wurde, kann mittels des in Listing 4.2 aufgeführten Befehls der eigentliche SCONE-Container für MariaDB – und damit die Datenbank innerhalb einer SGX-Enklave – gestartet werden.

```
# FSPF-Metadaten aus Datei lesen
FSPF_TAG=$(cat keytag | awk '{print $11}')
FSPF_KEY=$(cat keytag | awk '{print $9}')

# MariaDB-Container starten
docker run -d \
  --name mariadb-sgx \
  # Einbinden des SGX-Treibers (IAS-Version)
  --device /dev/isgx \
  # Verzeichnis für Datenbasis einbinden
  -v $PWD/database:/var/lib/mysql \
  # Konfigurationsdatei und Zertifikate einbinden
  -v $PWD/mariadb-server.cnf:/etc/my.cnf.d/mariadb-server.cnf↩
   ↪ \
  -v $PWD/db-certs:/etc/mysql-ssl \
  # FSPF einbinden
  -v $PWD/db.pb:/db.pb \
  # SCONE File System Shield aktivieren
  -e SCONE_FSPF_TAG=$FSPF_TAG \
  -e SCONE_FSPF_KEY=$FSPF_KEY \
  -e SCONE_FSPF=/db.pb \
  # Schreibzugriff auf geschützte Verzeichnisse durch MariaDB↩
   ↪ zulassen
  -e SCONE_FSPF_MUTABLE=1 \
    # SCONE-Image für MariaDB
    "sconecuratedimages/apps:mariadb-alpine"
```

Listing 4.2: Starten von MariaDB in SCONE

4.1.4.2 Lighttpd

Migration Anders als die DB wird der Webserver mit Hilfe von Graphene-SGX migriert (Version vom 10.09.2020[12]).
Dazu ist zuerst eine Installation dieses libOS nach der unter [Gra19a] vorliegenden Anleitung erforderlich, wobei an dieser Stelle statt eines gepatchten Kernels der Graphene-spezifische SGX-Treiber verwendet wird. Der Installationspfad wird dabei im Folgenden als `<GDir>` bezeichnet.
Graphene-SGX beinhaltet verschiedene Beispielprojekte unter `<GDir>/Examples/`, darunter auch eines für den Webserver Lighttpd. Folglich dienen die entsprechenden Konfigurationsdateien als Grundlage für diesen Migrationsschritt und werden – mit den notwendigen Modifikationen – zum Ausführen des WS in einer SGX-Enklave genutzt.

Zuerst müssen alle Lighttpd-Quelltexte heruntergeladen und die entsprechenden Programmdateien und Module kompiliert werden.
Entsprechend des Prinzips der minimalen TCB werden bei der Build-Konfiguration[13] mittels des `./configure`-Kommandos alle Funktionen deaktiviert, die nicht zwingend erforderlich sind. Andererseits müssen einige notwendige Module (z.B. `mod_openssl` für die Unterstützung von HTTPS) explizit spezifiziert werden, sodass beim Build-Prozess des Webservers das folgende Konfigurationskommando (siehe auch Listing B.4, Z. 46 f.) zum Einsatz kommt, wobei `<I>` das Installationsverzeichnis angibt.

```
./configure --prefix=<I> --without-pcre --without-zlib \
            --without-bzip2 --with-openssl
```

Anschließend wird eine Vorlage (`lighttpd.manifest.template`) verwendet, um das eigentliche *Manifest* sowie dessen SGX-Version (`lighttpd.manifest.sgx`) zu generieren und die Hashwerte aller in der Enklave benötigten und integritätsgeschützten Dateien zu berechnen.
Diese `trusted_files` sind in der Regel jene, die als unveränderlich angesehen werden und keine vertraulichen Informationen enthalten – in diesem Fall dynamische Bibliotheken sowie die Konfigurationsdateien des WS. Auch die Zertifikatsdatei `lighttpd.pem` wird in diesem Beispiel nur in Bezug auf ihre Integrität, nicht ihre Vertraulichkeit,

[12] Für die beispielhafte Migration des von der A_{MS} WordPress verwendeten Web- und Anwendungsservers wurde die Graphene-Version des GitHub-Branches *master* mit der Commit-ID *34b8eb1ec2b23d5538d8ede385eb1be50e29354d* verwendet.

[13] Der „Build-Prozess“ umfasst in der Softwareentwicklung alle Schritte (u.a. Kompilieren und Linken) zum Erstellen einer (ausführbaren) Binärdatei, ausgehend von einer Menge an Quellcodedateien (nach [Tec20]).

geschützt.
Alle weiteren Dateien und Verzeichnisse werden als `allowed_files` deklariert, da sich ihr Inhalt während der Laufzeit verändern kann. Nicht vertrauenswürdige Stellen außerhalb der Enklave können folglich uneingeschränkt auf diese zugreifen.
In der Praxis muss allerdings jegliche Zugriffsmöglichkeit auf vertrauliche Informationen von außerhalb der Secure Enclave unterbunden werden, z.B. durch Sealing als `protected_files`, wobei ein Zugriff nur von innerhalb der Enklave nach erfolgreicher Attestation und Secret Provisioning möglich ist. Dies betrifft insbesondere die WS-Zertifikate, Log- sowie die Dateien der WordPress-Anwendung.
Zusätzlich müssen für alle in der Enklave benötigten Dateien und Verzeichnisse des Hostsystems entsprechende `fs.mount`-Einträge in der *Manifest*-Vorlage hinzugefügt werden.
Um die Anwendung startbereit zu machen, erfolgt nach Erstellung der `*.template`-Datei abschließend die Ermittlung der Enklavensignatur, welche für die spätere Ausführung abgespeichert wird.
Der gesamte Prozess wird durch die Verwendung einer `Makefile`-Datei automatisiert, welche die Existenz der *Manifest*-Vorlage sowie aller Konfigurationsdateien des Webservers voraussetzt. Deren sowie die Inhalte des `Makefile` werden in Abschnitt B.2.2 im Anhang B angegeben.

Absicherung der Kommunikation Wie in Abschnitt 4.1.2 beschrieben, wird die HTTP-Verbindung zwischen Webserver und Nutzer durch Verwendung des TLS-Protokolls kryptografisch abgesichert.
Dazu muss (nach [gst20]) das Lighttpd-Modul `mod_openssl` aktiviert und konfiguriert werden – beispielsweise mit den in Listing B.10 aufgeführten Einstellungen. Zusätzlich benötigt der WS eine X.509-Zertifikatskette mit seinem privat-öffentlichen Schlüsselpaar. Zu dessen Erstellung kann z.B. das Paket „OpenSSL“ mit dem folgenden Kommandozeilenbefehl verwendet werden.

```
openssl req -new -x509 -keyout lighttpd.pem -out lighttpd.pem \
                                -days 730 -nodes
```

Die FastCGI-Kommunikation zwischen Web- und Anwendungsserver bedarf in Produktivumgebungen ebenfalls des kryptografischen Schutzes.
Da dieses Protokoll jedoch keine Option zum verschlüsselten Datenaustausch und das Library OS Graphene-SGX keine Möglichkeit bietet, diese nachträglich hinzuzufügen[14],

[14]Möglich wäre z.B. eine transparente Verschlüsselung allen TCP-Datenverkehrs, ähnlich wie bei der SCONE-Funktion des *Network Shields* [Arn+16, S. 694 f.], welche mit der „Standard“-Lizenz dieser Shielding Layer verfügbar ist.

muss der Kommunikationskanal C(Lighttpd, PHP-FPM) im Rahmen dieser Arbeit ungeschützt bleiben.

Ausführen des Webservers in Graphene-SGX Zur Vorbereitung der Graphene-Ausführungsumgebung und anschließendem Starten des Webservers innerhalb dieses libOS – und damit in einer SGX-Enklave – können die in Listing 4.3 dargestellten Befehle verwendet werden. Dabei wird vorausgesetzt, dass das Verzeichnis `<GDir>/Examples/lighttpd/` folgende Dateien beinhaltet: die fünf, in den Listings B.6 bis B.10 dargestellten, Konfigurationsdateien, die *Manifest*-Vorlage und das `Makefile` sowie die generierte Zertifikatsdatei.

```
1 cd <GDir>/Examples/lighttpd
2 # Kompilieren des WS und Generieren aller Graphene-Metadaten
3 make SGX=1
4 # Starten des Webservers in Graphene-SGX
5 SGX=1 make start-graphene-server
```

Listing 4.3: Starten von Lighttpd in Graphene-SGX

4.1.4.3 PHP-FPM

Migration Auch für die Ausführung des AS in SGX-Enklaven wird eine korrekte Installation des Library OS vorausgesetzt. Da Graphene allerdings kein Beispielprojekt für PHP bereitstellt, wird ein neues Verzeichnis unter (`<GDir>/Examples/php/`) erstellt.
Nach dem Herunterladen aller notwendigen Quelltexte erfolgt die Konfiguration des Kompiliervorgangs für PHP-FPM. Zum Ausführen der WordPress-Skripte müssen dabei 21 verschiedene, in Tabelle B.1 (Anhang B) aufgelistete, Erweiterungen aktiviert werden [Wor20e]. Durch den folgenden `./configure`-Befehl (siehe auch Listing B.12, Z. 18 ff.) werden diese als statische Bibliotheken kompiliert und in die Programmdatei `php-fpm` eingebunden[15].

```
./configure --disable-phpdbg --disable-pdo --without-pdo-sqlite \
--without-sqlite3 --with-curl --enable-exif --enable-ftp --enable-gd \
  --with-sodium --enable-mbstring --with-mysqli --with-openssl \
      --enable-sockets --with-zip --with-zlib --enable-fpm
```

[15] Zum erfolgreichen Kompilieren des PHP-Interpreters mit den angegebenen Erweiterungen müssen die entsprechenden Bibliotheken im System verfügbar sein. Im Fall von Ubuntu ist dazu die Installation der folgenden Pakete notwendig: `libcurl4-openssl-dev`, `libxml2-dev`, `libssl-dev`, `zlib1g-dev`, `libonig-dev`, `libsodium-dev`, `libpng-dev` und `libzip-dev`.

Nach erfolgreichem Abschluss des Build-Prozesses folgt die Erstellung der *Manifest*-Vorlage für PHP-FPM.
Dabei werden alle benötigten dynamischen Bibliotheken sowie die Konfigurationsdateien als `trusted_files` deklariert, weil sich ihr Inhalt nicht ändert und damit Graphene-SGX ihre Integrität vor dem Laden in eine Enklave sicherstellen kann.
Alle weiteren Dateien und Verzeichnisse, die der AS innerhalb der Enklave benötigt (Log-, WordPress- und Systemdateien), werden an dieser Stelle als `allowed_files` in die Vorlage eingetragen, da sie veränderliche Informationen enthalten bzw. von PHP selbst modifiziert werden können (z.B. Änderung von Konfigurationen oder Hinzufügen bzw. Entfernen von Dateien wie Uploads).
Wie bereits bei der Migration des Webservers angeführt, müssen im Produktiveinsatz jedoch jegliche vertrauliche Inhalte durch die Verwendung von `protected_files` o.Ä. vor Zugriffen von außerhalb der Enklave geschützt werden.
Die fertiggestellte *Manifest*-Vorlage ist in Listing B.13 enthalten. Wieder wird der gesamte Prozess des Quelltext-Downloads, Kompilierens und Generierens der benötigten Graphene-Dateien durch die Verwendung eines `Makefile` (siehe Listing B.12) automatisiert.

Für die Ausführung des Anwendungsservers werden drei Konfigurationsdateien benötigt – je eine für den PHP-FPM-Prozess selbst (`php-fpm.conf`), den für WordPress eingesetzten Workerpool (`www.conf`) sowie jeden einzelnen PHP-FastCGI-Prozess (`php.ini`).
In der Pool-Definition kann die Art der Prozessverwaltung durch den FPM bestimmt werden. Die Einstellung *dynamic* ermöglicht dabei eine flexible und bedarfsgerechte Erzeugung und Beendigung von PHP-Prozessen. Im Praxisversuch wurde allerdings festgestellt, dass der Prozessmanager innerhalb des Library OS Graphene-SGX die aktuelle Auslastung der Kindprozesse nicht korrekt bestimmen und damit diese Option nicht umsetzen kann. Aus diesem Grund muss für den WordPress-Anwendungsserver stattdessen ein Workerpool vom Typ *static* gewählt werden, welcher vier, ständig verfügbare, PHP-Prozessen enthält (siehe Listing B.15, Z. 19).
Der Inhalt aller drei Konfigurationsdateien wird im Anhang B, Abschnitt B.2.3 aufgeführt.

Absicherung der Kommunikation Wie in Abschnitt 4.1.4.2 aufgezeigt, bietet das FastCGI-Protokoll keine Möglichkeit zur Verschlüsselung, weshalb die Schnittstelle zwischen WS und AS im Rahmen dieser Arbeit ungeschützt bleibt.
Die Kommunikation mit der DB kann dagegen verschlüsselt erfolgen. Nach der Konfiguration des MariaDB-Servers (siehe Abschnitt 4.1.4.1) ist dazu eine Anpassung in der WordPress-Datei `wp-config.php` notwendig. So muss die Zeile 90 aus Listing 4.4

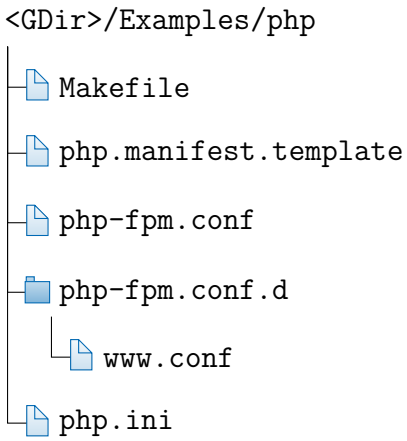
am Ende der Konfigurationsdatei hinzugefügt werden, jedoch zwingend vor der letzten Zeile, welche zum Einbinden weiterer Einstellungen verwendet wird.

```
82 /* That's all, stop editing! Happy publishing. */
83
84 /** Absolute path to the WordPress directory. */
85 if ( ! defined( 'ABSPATH' ) ) {
86   define( 'ABSPATH', __DIR__ . '/' );
87 }
88
89 // Diese Zeile für verschlüsselte Kommunikation mit der DB ↩
     ↪hinzufügen!
90 define('MYSQL_CLIENT_FLAGS', MYSQLI_CLIENT_SSL);
91
92 /** Sets up WordPress vars and included files. */
93 require_once ABSPATH . 'wp-settings.php';
```

Listing 4.4: WordPress-Konfiguration für die Verschlüsselung des DB-Protokolls (Datei `wp-config.php`)

Durch diese Einstellung wird die Kommunikation C(PHP-FPM, MariaDB) nach einem Neustart der Webanwendung mittels TLS geschützt, wobei diese Verschlüsselung an beiden Enden der Kommunikation innerhalb einer Enklave terminiert wird. So lange keine andere Stelle Zugriff auf die zur Verschlüsselung verwendeten privaten Schlüssel bekommt, können die auf diesem Weg übertragenen Daten somit (gemäß dem SGX zugrundeliegenden Bedrohungsmodell) als sicher angesehen werden.

Ausführen des Anwendungsservers in Graphene-SGX Unter Voraussetzung der folgenden Verzeichnisstruktur kann schließlich, mit Hilfe von äquivalenten `make`-Befehlen wie für den Webserver (siehe Listing 4.3), die Ausführung von PHP-FPM im Library OS vorbereitet und gestartet werden.



4.2 Evaluation der Performanzreduktion

Durch die Durchführung der in den Abschnitten 4.1.1 bis 4.1.4 beschriebenen Migrationsschritte konnte die komplette Multi Service Application „WordPress“ in mehrere SGX-Enklaven migriert werden. Wie in Abschnitt 2.1.7 erläutert, bringt die mit Hilfe der Software Guard Extensions erhöhte Sicherheit Einschränkungen in Bezug auf die Ausführungsgeschwindigkeit und Robustheit der entsprechenden Anwendung mit sich. Um eine praktische Einschätzung dieser Auswirkungen treffen zu können, wurden anhand des WordPress-Beispiels verschiedene Tests und Benchmarks zur Quantisierung der Performanzeinbußen durchgeführt. In diesem Abschnitt wird sowohl die Testmethodik als auch deren Ergebnisse vorgestellt und diese ausgewertet.

4.2.1 Testumgebung

Zur Sicherstellung nachvollziehbarer Ergebnisse werden die relevanten Eigenschaften der für die Performanzbetrachtungen eingesetzten Testumgebung im Folgenden aufgeführt. Neben der Hardware des verwendeten Rechners werden dabei auch die zum Testen genutzten Werkzeuge und Programme sowie die Softwareumgebung der WordPress-Installation beschrieben.

Eigenschaften des Testrechners Für die Performanztests wurde ein SGX-fähiger Rechner eingesetzt, der die in Tabelle 4.1 aufgeführten Eigenschaften aufweist.

Modellbezeichnung	Lenovo ThinkPad T470s
CPU	Intel Core i7-7600U (2 physische, 4 logische Kerne mit je 2,80 GHz)
Architektur	64 Bit
RAM	20 GB (DDR4)
Betriebssystem	Ubuntu 18.04 LTS
Kernelversion	Linux 4.15.0-109-generic
SGX-Treiber-Version	2.6.0
SGX-Modus	*Debug*

Tabelle 4.1: Eigenschaften des Testgeräts für WordPress in Intel SGX

Verwendete Testsoftware Um die Ausführungsgeschwindigkeit der Webanwendung zu quantifizieren, kommen drei verschiedene Werkzeuge zum Einsatz, die in diesem Abschnitt kurz vorgestellt werden sollen.

Für Testszenarien mit lesendem Zugriff auf WordPress-Beiträge wird das Benchmark-Programm „Siege“ [Ful12] in der Version 4.0.7 eingesetzt.
Zur Einschätzung der durch SGX verursachten Performanzreduktion soll die Zeit vom Übermitteln einer einzelnen Anfrage an den Webserver bis zum Fertigstellen des Downloads aller zum jeweiligen Blogpost gehörigen Dateien gemessen werden (dazu gehören jegliche Berechnungen des AS sowie Zugriffe auf die DB). Da Siege diese spezifische Metrik nicht berechnet, sind zuvor jedoch Anpassungen an dessen Quellcode notwendig.
Dazu wird das Programm aus seinem GitHub-Repository [Ful20] heruntergeladen, die Datei `siege/src/main.c` entsprechend der in Listing C.1 enthaltenen Änderungen[16] modifiziert und anschließend – nach der Installationsanleitung im Siege-Verzeichnis `siege/INSTALL` – kompiliert. Zukünftig wird die beschriebene Metrik bei Beendigung des Benchmark-Programms als `request_time` in Sekunden ausgegeben.
Weiterhin wurde bei den mittels Siege durchgeführten Performanztests die in Listing C.2 aufgelistete Konfigurationsdatei verwendet.

Die Zeit für die Durchführung eines schreibenden Zugriffs auf die WordPress-Anwendung wird stattdessen durch das HTTP-Client-Programm „cURL“ [Hax20] in Kombination mit automatisierenden Shellskripten gemessen. Auch beim Herunterladen größerer Dateien wird diese Software eingesetzt, weil Siege den entsprechenden Speicheranforderungen nicht gerecht wird.

[16] Konkret müssen in der Siege-Quelltext-Datei die Zeilen 528, 543 und 573 hinzugefügt werden.

Die benötigte Variabilität in Bezug auf die Anzahl paralleler Anfragen sowie der Größe von Dateien und Inhalten wird mittels der in den Listings C.3 und C.4 aufgeführten Bash-Skripte erreicht.

Zuletzt wird das Analyseprogramm für Netzwerkverkehr „Wireshark“ [Wir20] zur Aufzeichnung von Paketen eingesetzt.
Ist eine Automatisierung mit Hilfe der beiden anderen Werkzeuge nicht möglich, wird auf dieses Programm zurückgegriffen und die Dauer einer HTTP-Verbindung durch Auswertung der protokollierten Zeitstempel manuell berechnet. Zur Erfassung von Netzwerkpaketen zwischen Client und Webserver sowie zwischen WS und AS wird die Schnittstelle für den lokalen Rechner (unter Ubuntu *lo*), für die Kommunikation zwischen PHP-FPM und MariaDB dagegen die des Docker-Dienstes (*docker0*), als „Capture Interface“ verwendet.

WordPress-Installationen Um aussagekräftige Ergebnisse zu erhalten, ist es notwendig, jeden Test sowohl mit der in Abschnitt 4.1 entwickelten SGX-Variante als auch mit einer WordPress-Installation ohne Enklaven durchzuführen. Dabei sollten diese beiden Ausführungsumgebungen so weit wie möglich aneinander angeglichen werden, um eine Performanzbeeinflussung durch Störquellen abseits von SGX zu minimieren. Da der Aufbau der Enklaven verwendenden Webanwendung bereits beschrieben wurde, soll an dieser Stelle die alternative Variante einer WordPress-Installation ohne SGX vorgestellt werden. Letztere wird nachfolgend als „nativ“ bezeichnet.

Webserver Als WS kommt ebenfalls Lighttpd – mit der gleichen Installation wie in Graphene-SGX – zum Einsatz. Dazu wird das Verzeichnis `<GDir>/Examples/lighttpd/` sowie das darin enthaltene und in Listing B.4 (Anhang B) dargestellte `Makefile` wiederverwendet. Mit dem folgenden Befehl kann der Webserver schließlich unter Verwendung einer alternativen Hauptkonfigurationsdatei (`lighttpd-native-server.conf` aus Listing B.11 statt `lighttpd.conf`) gestartet werden.

```
make start-native-server
```

Applikationsserver Auch in der nativen Variante wird die PHP-FPM-Anwendung aus dem Graphene-Verzeichnis `<GDir>/Examples/php/` verwendet. Weiterhin kommen auch die im Library OS genutzten Konfigurationsdateien zum Einsatz, wobei durch das entsprechende `Makefile` bei der Ausführung des AS lediglich Anpassungen bezüglich des Speicherortes der Logdateien vorgenommen werden (siehe Listing B.12, Z. 90 ff.). Zum Starten des PHP-FPM wird das gleiche `make`-Kommando wie für Lighttpd ausgeführt.

Datenbankserver Da MariaDB in der SGX-Variante als Docker-Container ausgeführt wird, muss sich der DB-Server für eine bessere Vergleichbarkeit auch beim nativen WordPress innerhalb eines Containers befinden.
Zu diesem Zweck wird das Docker-Image `yobasystems/alpine-mariadb` in der Version 10.4.12 verwendet, da es annähernd die gleichen Softwareversionen wie das SCONE-Image[17] beinhaltet. Analog zu letzterem wird der DB-Container der nativen Installation durch das in Listing 4.5 aufgeführte Kommando gestartet.

```
1 docker run -d --name mariadb-native \
2   -v \$PWD/database-native:/var/lib/mysql \
3   -v \$PWD/mariadb-server.cnf:/etc/my.cnf.d/mariadb-server.↩
    ↪cnf \
4   -v \$PWD/db-certs:/etc/mysql-ssl \
5   -e MYSQL_ROOT_PASSWORD="<root_PW>" \
6   -e MYSQL_DATABASE="wordpress" \
7   -e MYSQL_USER="wp" \
8   -e MYSQL_PASSWORD="wordpress" \
9     "yobasystems/alpine-mariadb:10.4.12"
```

Listing 4.5: Startbefehl für den nativen MariaDB-Server

In beiden Fällen werden alle drei WordPress-Dienste jeweils auf dem gleichen Rechner gestartet und kommunizieren demnach über die lokale (*lo*) bzw. Docker- (*docker0*) Netzwerkschnittstelle. Auch die jeweils verwendeten Testwerkzeuge kommen auf dem gleichen Gerät zum Einsatz.

4.2.2 Testverfahren

Um eine fundierte Aussage über den Einfluss der Verwendung von Secure Enclaves auf die Ausführungsumgebung von WordPress treffen zu können, erfolgte im Rahmen dieser Arbeit die Entwicklung und Durchführung diverser Testverfahren. Die dabei gewonnenen Erkenntnisse können bezüglich der Migration von Multi Service Applications generalisiert werden.
Insbesondere wurde Priorität auf die praktische Relevanz der Ergebnisse in Bezug auf die Verwendung von WordPress als Werkzeug zum Veröffentlichen und Bereitstellen von Blogbeiträgen gelegt. Aus diesem Grund handelt es sich hierbei um sog. „Makro-

[17] Beide Container-Images beinhalten MariaDB in der Version 10.4.12 und nutzen „Alpine Linux“ als Container-OS. Der einzige Unterschied liegt in dessen Version (yobasystems: 3.11.3, SCONE: 3.11.6).

Benchmarks“, die aus Sicht des Endnutzers durchgeführt werden (für weitere Informationen siehe z.B. [WK00]). In den Performanztests der $A_{MS,p}$ kommen folglich die folgenden Testszenarien zum Einsatz, wobei in jedem Anwendungsfall die (Datei-) Größe bzw. die Anzahl paralleler Zugriffe variiert wird.

1. Aufrufen eines veröffentlichten Blogposts
2. Herunterladen einer von WordPress bereitgestellten Datei
3. Veröffentlichen eines Kommentars zu einem vorhandenen Post
4. Hochladen einer Datei

4.2.2.1 Aufrufen eines Blogposts

Der Download eines mittels WordPress veröffentlichten Blogbeitrags erfolgt durch eine HTTP-GET-Anfrage. In der Standardeinstellung werden solche Beiträge anhand ihrer Post-ID identifiziert, weshalb der entsprechende Inhalt bei Verwendung der beschriebenen WordPress-Installation unter der folgenden URL verfügbar ist.

`https://localhost:8443/?p=<PostID>`

Zum Testen der Ausführungsgeschwindigkeit der Webanwendung kommt (nach Erstellung eines kleinen, beispielhaften WordPress-Posts) das Programm „Siege“ zum Einsatz. Der Webclient muss für diesen Beitrag ca. 430 kiB Daten herunterladen.
Durch Ausführung der Kommandos aus Listing 4.6 wird die für die Bereitstellung des angeforderten Blogbeitrags benötigte Zeit sowie weitere Metriken in der Datei `results_post_download.txt` gespeichert.
Zur Eliminierung durch kurzfristige Störungen verursachter Spitzen wird der Test jeweils 20x durchgeführt und die Ergebnisse gemittelt. Außerdem wird die Bereitstellungszeit mehrmals mit unterschiedlichen parallelen Nutzerzahlen[18] gemessen, um die Skalierbarkeit der Webanwendung bei steigenden Anforderungen zu ermitteln.

```
1 # Angabe der Post-ID
2 POST_ID=<Post-ID>
3 # Verwendung der Siege-Konfigurationsdatei "siegerc"
4 SIEGERC=./siegerc
5 # 20 Wiederholungen je Test
```

[18] Alle Performanztests mit gleichzeitigem Zugriff wurden jeweils mit den folgenden parallelen Nutzerzahlen durchgeführt: 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

```
6  REPEATS=20
7  # Speicherung der Ergebnisse in die Datei
8  # "results_post_download.txt"
9  RESULTS_FILE=./results_post_download.txt
10
11 # Test für einen einzelnen parallelen Zugriff
12 echo "#users=1" > $RESULTS_FILE
13 siege --rc=$SIEGERC --concurrent=1 --reps=$REPEATS --delay=0 ↩
     ↪-j https://localhost:8443/?p=$POST_ID >> $RESULTS_FILE
14 # Test für 10 bis 100 parallele Zugriffe
15 for ((i=10; i <= 100; i+=10)); do
16   echo "#users=$i" >> $RESULTS_FILE
17   siege --rc=$SIEGERC --concurrent=$i --reps=$REPEATS --↩
     ↪delay=0 -j https://localhost:8443/?p=$POST_ID >> ↩
     ↪$RESULTS_FILE
18 done
```

Listing 4.6: Zeitmessung der Post-Downloads mittels Siege

Zur weiteren Verwendung werden die relevanten Messdaten aus der generierten Textdatei durch das Python-Skript `WordPress-Tests/Siege/parse-siege.py`, welches auf dem dieser Arbeit beiliegenden Datenträger enthalten ist, extrahiert und in eine CSV-Datei geschrieben.

4.2.2.2 Datei-Download

Neben dem Betrachten eines Blogbeitrags kann der Nutzer einer WordPress-Seite auch bereitgestellte Dateien (z.B. Fotos oder Videos) herunterladen.
Um die Geschwindigkeit zu messen, mit der unterschiedlich große Dateien durch den Webserver Lighttpd unter SGX- und Nicht-SGX-Bedingungen ausgeliefert werden, kommt das Kommandozeilenprogramm cURL zum Einsatz, da es – im Gegensatz zu Siege – mit größeren Dateien umgehen kann und in diesem Fall keine Parallelität benötigt wird. Der Zugriff auf solche (Medien-) Dateien erfolgt in der Standardeinstellung über eine URL, welche den Zeitstempel des Hochladedatums sowie den Dateinamen beinhaltet:

`https://localhost:8443/wp-content/uploads/<UploadJahr>/<UploadMonat>/<Dateiname>`

Bei diesem Test werden neun Dateien mit zufälligem textuellen Inhalt und jeweils unterschiedlichen Dateigrößen verwendet[19]. Auch der Download-Test wird 20x wiederholt und für jede Dateigröße der Mittelwert der gemessenen Downloadzeiten und -geschwindigkeiten gebildet.
Zur vereinfachten Ausführung des Benchmarks wurde ein (Bash-) Shellskript entwickelt, welches mittels cURL die für das Herunterladen benötigte Zeit sowie die durchschnittliche Datentransferrate bestimmt und die jeweiligen Mittelwerte berechnet. Dazu wird eine Textdatei mit den herunterzuladenden Dateinamen benötigt (siehe Listing C.5 im Anhang C). Der Inhalt des Shellskriptes selbst wird in Listing C.3 dargestellt. Durch das folgende Kommando wird die Messung schließlich gestartet und deren Ergebnisse in die CSV-Datei `results_media_download.csv` gespeichert.

```
./download-bench.sh
```

4.2.2.3 Veröffentlichen eines Kommentars

Ein Nutzer mit Zugriff auf einen existierenden WordPress-Beitrag kann (bei entsprechend konfigurierter WebApp) einen oder mehrere Kommentare unter diesem verfassen. Bei einem solchen schreibenden Zugriff wird eine HTTP-POST-Anfrage an den Webserver (und nachfolgend den AS) gestellt, der den Kommentar in der verknüpften Datenbank abspeichert und den Nutzer über Erfolg oder Fehlschlag informiert.
Da bei der Erstellung eines neuen Blogbeitrags ein analoges Prozedere – mit Beteiligung der gleichen A_{MS}-Dienste – angewendet wird, sind die in diesem Testszenario gewonnenen Erkenntnisse auch auf jenen Anwendungsfall übertragbar.
Um die für den gesamten Vorgang benötigte Zeit automatisiert zu messen, wird die von WordPress bereitgestellte Programmierschnittstelle zum Veröffentlichen der Kommentare verwendet, welche als Representational State Transfer (kurz REST-) API vorliegt[20] [Wor20d]. Mittels eines weiteren, in Listing C.4 dargestellten, Shellskriptes verbindet sich cURL mit dem API-Endpunkt und stellt alle Informationen über den jeweiligen Kommentar bereit, sodass auf diese Weise eine programmatische Variation in Bezug auf Kommentarumfang und gleichzeitig zugreifende Nutzer möglich ist[21]. Vor

[19] Zum Testen wurden mit Hilfe des Kommandozeilenprogramms OpenSSL zufällige Dateien mit den folgenden Größen generiert: 512 Byte, 1 MiB, 16 MiB, 64 MiB, 512 MiB, 1 GiB, 2 GiB, 4 GiB, 8 GiB

[20] Um sich über die Kommandozeile gegenüber der API authentifizieren zu können, ist die Installation und Aktivierung des WordPress-Plugins „JSON Basic Authentication“ [Wor17] erforderlich.

[21] Der Test wird jeweils für zufällige Kommentartexte mit 2 (minimale Länge), 5.000, 30.000 und 65.525 (maximale Länge) Zeichen sowie für die im Abschnitt 4.2.2.1 aufgeführten parallelen Nutzerzahlen durchgeführt.

der Ausführung müssen in diesem Skript allerdings die Zugangsdaten des WordPress-Admin-Nutzers als `<Nutzername>:<Passwort>` in Zeile 3 angegeben werden.
Damit die erstellten Mitteilungen tatsächlich auf der Seite eines Beitrages veröffentlicht werden, ist außerdem eine Anpassung der Standardeinstellung zur Kommentarfreigabe erforderlich (siehe [Wor20a]).
Zur Mittelwertbildung wird jede Anfrage fünfmal wiederholt, wobei für jeden Kommentar ein anderer Titel gewählt werden muss, da WordPress keine identischen Überschriften akzeptiert.
Die folgenden Befehle führen diesen Test durch und speichern die Ergebnisse in je einer CSV-Datei für die verschiedenen Nutzerzahlen, wobei die Post-ID eines existierenden Blogbeitrags angegeben wird und die Zeichenlängen der zu erstellenden Kommentare aus der Datei `comment-lengths.txt` (siehe Listing C.6) eingelesen werden.

```
1 # Test für einen einzelnen Zugriff
2 ./comment-bench.sh -o results_comments_1.csv \
3   -L comment-lengths.txt -p <PostID>
4 # Test für 10 bis 100 parallele Zugriffe
5 for ((i=10; i <= 100; i+=10)); do
6   ./comment-bench.sh -o results_comments_$i.csv \
7     -L comment-lengths.txt -n $i -p <PostID>
8 done
```

Listing 4.7: Zeitmessung der Kommentar-Erstellung mittels cURL

4.2.2.4 Datei-Upload

Zuletzt soll die für das Hochladen von (Medien-) Dateien unter SGX benötigte Zeit gemessen und mit der eines Standard-WordPress-Szenarios verglichen werden.
Da diese Aktion in der Regel nur vom Autor eines Blogs durchgeführt wird und andere Nutzer keine Berechtigung zum Hochladen von Dateien besitzen, wird bei diesem Test auf die Parallelität verzichtet. Um den Einfluss der Dateigröße auf die Performanz zu quantifizieren, kommen die gleichen Testdateien mit zufälligem Inhalt wie in Abschnitt 4.2.2.2 zum Einsatz.
Die für das Verfassen von Kommentaren verwendete REST-API bietet auch die Möglichkeit, Dateien hochzuladen. Im Gegensatz zum Upload mit Hilfe der WordPress-Weboberfläche wird eine über die API hochgeladene Datei allerdings vom Server nicht direkt auf die Festplatte geschrieben, sondern zuvor komplett im Arbeitsspeicher vorgehalten.
Da dies – vor allem für größere Dateien – nicht praktikabel ist und insbesondere in SGX-

Enklaven nur wenig Speicher zur Verfügung steht, kann die Programmierschnittstelle für dieses Szenario nicht zum Einsatz kommen. Stattdessen muss der Test manuell durchgeführt, mittels Wireshark protokolliert und die benötigte Zeit im Anschluss berechnet werden. Auch auf eine Wiederholung des Experiments und die Berechnung des Mittelwertes wird deshalb verzichtet.

Um die Zeit für das Hochladen messen zu können, müssen die Header der von und zu WordPress verschickten Pakete analysiert werden. Da die für die Kommunikation zwischen Client und Webserver aktivierte TLS-Verschlüsselung dies verhindert, muss sie zur Durchführung dieses Tests temporär deaktiviert werden[22].
Anschließend wird Wireshark gestartet und der Aufzeichnungsvorgang durch Auswahl der Netzwerkschnittstelle *Loopback: lo* begonnen. Zur besseren Übersicht kommt zusätzlich der Analysefilter `tcp.port == 8080` zum Einsatz.
Das Hochladen der Dateien selbst erfolgt mittels der WordPress-Seite zur Administration, welche (nach Umstellung auf HTTP) unter der folgenden URL erreichbar ist.

```
http://localhost:8080/wp-admin/media-new.php
```

Nach Auswahl der Option Browser Downloader kann eine hochzuladende Datei ausgewählt und der Upload-Vorgang anschließend gestartet werden.
Wurde dieser abgeschlossen, leitet WordPress den Browser auf die Übersichtsseite mit allen hochgeladenen Dateien weiter und der Protokollmitschnitt durch Wireshark kann beendet werden. Anschließend wird die für das Hochladen benötigte Zeit durch die Bildung der Differenz zwischen den Zeitstempeln (1.) des Paketes mit der HTTP-POST-Anfrage auf die Ressource `/wp-admin/media-new.php` und (2.) jenem mit der Antwort des Webservers über die Beendigung des Vorgangs und Weiterleitung auf die Übersicht (HTTP-Antwortcode: „302 Found“) berechnet.

4.2.3 Testergebnisse

In diesem Abschnitt werden die Ergebnisse der beschriebenen Testszenarien grafisch dargestellt und ausgewertet.

[22] Da die eingesetzten Lighttpd-Konfigurationsdateien (siehe Abschnitt B.2.2) sowohl die Verbindung mittels HTTP als auch HTTPS erlauben, müssen zur Verwendung der unsicheren Protokollvariante lediglich Einstellungen in der Webanwendung geändert werden.
Dazu wird die Administrations-Oberfläche unter `https://localhost:8443/wp-admin/options-general.php` aufgerufen und in den Textfeldern der WordPress Address (URL) sowie der Site Address (URL) die neue URL `http://localhost:8080` eingetragen. Fortan kann WordPress nur noch unter dieser Adresse erreicht werden.

4.2.3.1 Aufruf eines Blogposts

Im ersten Szenario wurde mittels Siege die Reaktionszeit von WordPress als Webanwendung gemessen. Um den Einfluss der verwendeten SGX-Enklaven zu bestimmen, wurde diese Zeit sowohl für die migrierte als auch für die native Installation ermittelt. Zusätzlich erfolgte der gleiche Performanztest in drei weiteren WordPress-Konfigurationen, in denen jeweils nur ein einzelner Dienst der Multi Service Application in SGX-Enklaven migriert wurde, die beiden verbleibenden jedoch nativ ausgeführt werden. Dadurch ist es möglich, herauszufinden, welche Verringerung der Gesamtperformanz die Migration der einzelnen Teilanwendungen mit sich bringt.
Die Ergebnisse dieser fünf unterschiedlichen Zeitmessungen werden in Abbildung 4.3 dargestellt.

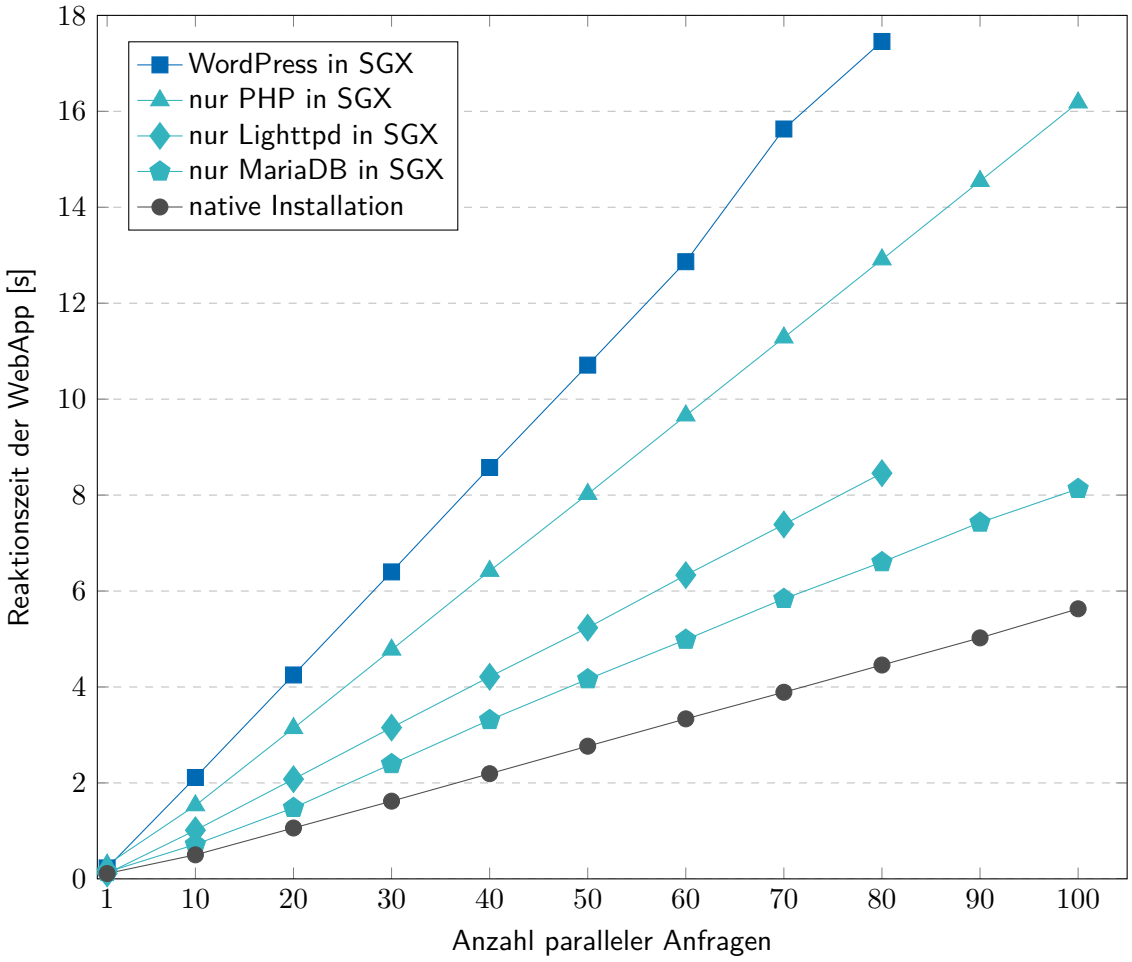


Abbildung 4.3: Einfluss paralleler Zugriffe auf die Reaktionszeit von WordPress

Durch die grafische Veranschaulichung wird deutlich, dass in jedem Testfall die Reaktionszeit der Webanwendung in einem (annähernd) linearen Zusammenhang zur Anzahl der parallelen Zugriffe steht.

Wie erwartet, benötigt das komplett in SGX ausgeführte WordPress jeweils am meisten Zeit für seine Berechnungen. Bei genauerer Untersuchung kann festgestellt werden, dass diese Variante eine ca. vierfache Reaktionszeit gegenüber der nativen Installation in allen Parallelitätsstufen aufweist.
Weiterhin fällt das Fehlen der Reaktionszeiten für 90 und 100 parallele Anfragen an die migrierte A_{MS} auf. Dies kann durch das Abstürzen des Webservers in Graphene-SGX bei mehr als 80 gleichzeitigen Zugriffen erklärt werden. Wird der Siege-Benchmark mit 90 Nutzern gestartet, erfolgt nach kurzer Zeit eine Terminierung von Lighttpd durch das Library OS mit dem Hinweis auf das Erreichen des in dessen Quellcode gesetzten Speicherlimits. Aus diesem Grund kann das Fehlen der letzten zwei Zeitwerte auch bei der Messreihe beobachtet werden, für die nur der Webserver migriert wurde.
Zuletzt kann festgestellt werden, dass die größte zeitliche Verzögerung durch den Anwendungsserver PHP-FPM verursacht wird, während der WS und die DB einen geringeren Einfluss auf die Gesamtperformanz ausüben. Diese Beobachtung ist nachvollziehbar, da der AS die eigentliche Anwendungslogik der WebApp beinhaltet und viele (Rechen-) Operationen ausführt.

4.2.3.2 Datei-Download

In diesem Testszenario erfolgte die Messung der für das Herunterladen von Dateien unterschiedlicher Größe benötigten Zeit mittels cURL. Die Ergebnisse für den SGX-Aufbau sowie die native Variante von WordPress werden in Abbildung 4.4 grafisch dargestellt.

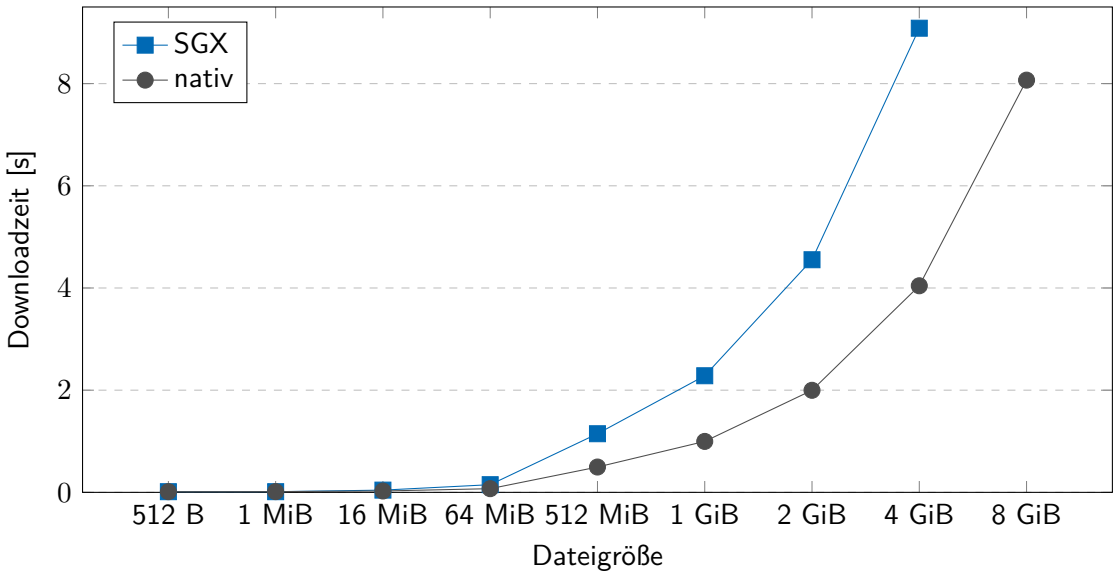


Abbildung 4.4: Einfluss der Dateigröße auf die Download-Zeit von WordPress

Mit Blick auf die beiden Kurven kann festgestellt werden, dass sich die Zeit zum Herunterladen (vor allem bei den größeren Dateien ab 512 MiB) proportional zur Dateigröße verhält. Dies gilt sowohl für die native als auch für die SGX-Installation. In letzterem Fall benötigt der Webserver generell etwa die doppelte Zeit wie bei der direkten Ausführung auf dem Hostsystem. Im Vergleich zum ersten Testszenario fällt die Verringerung der Performanz durch den Einsatz von Secure Enclaves beim Datei-Download demnach geringer aus. Dies kann damit erklärt werden, dass an einem solchen Vorgang nicht die gesamte Webanwendung, sondern nur der Webserver beteiligt ist, weil statische Inhalte – wie die generierten Textdateien – nicht an den AS zur Verarbeitung weitergereicht werden müssen.
Zuletzt fällt auf, dass im SGX-Aufbau kein Messwert für die Datei mit einer Größe von 8 GiB vorhanden ist. Dieser konnte durch den Benchmark nicht bestimmt werden, da der Download-Vorgang durch den Webserver (mittels eines `shutdown()`-Systemcalls) abgebrochen wird, bevor die Datei vollständig heruntergeladen wurde.
Die Terminierung erfolgt hierbei jeweils nach exakt 4 GiB übertragenen Daten. Aus diesem Grund konnte auch die 4-GiB-Datei nicht vollständig heruntergeladen werden – da die beim Download verwendeten Protokollheader vor dem eigentlichen Dateiinhalt übermittelt werden, fehlen dem Client die letzten 222 Byte der angeforderten Datei. Auch eine Erhöhung des maximal durch die Lighttpd-Enklave nutzbaren Speichers (siehe Abschnitt 4.2.3.4) führt nicht zum erfolgreichen Download dieser beiden Dateien.

4.2.3.3 Veröffentlichen eines Kommentars

Zum Test der Auswirkungen von Intel SGX auf die Verarbeitungsgeschwindigkeit für schreibende WordPress-Zugriffe wurden Kommentare unter einem bestehenden Blogbeitrag veröffentlicht.
Dabei erfolgte die Messung der Reaktionszeit vom Absenden der API-Anfrage bis zum Empfangen der Antwort mittels cURL. Dieser Test wurde sowohl mit der SGX-Installation als auch mit der nativen Variante der Webanwendung, für je vier verschiedene Kommentarlängen und mit unterschiedlicher Anzahl paralleler Zugriffe durchgeführt. In Abbildung 4.5 werden die entsprechenden Ergebnisse veranschaulicht, wobei die vier verschiedenen Zeichenlängen der veröffentlichten Kommentare jeweils durch vier Farbabstufungen dargestellt werden (je dunkler die Farbe, desto länger der Text).

Anhand der Testergebnisse kann festgestellt werden, dass die Reaktionszeit der Webanwendung bei der Veröffentlichung eines Kommentars mit höherem parallelen Nutzeraufkommen in jedem Fall ansteigt. Bei der nativen Installation von WordPress ist dieser Anstieg jedoch signifikant geringer als bei der Nutzung von SGX.

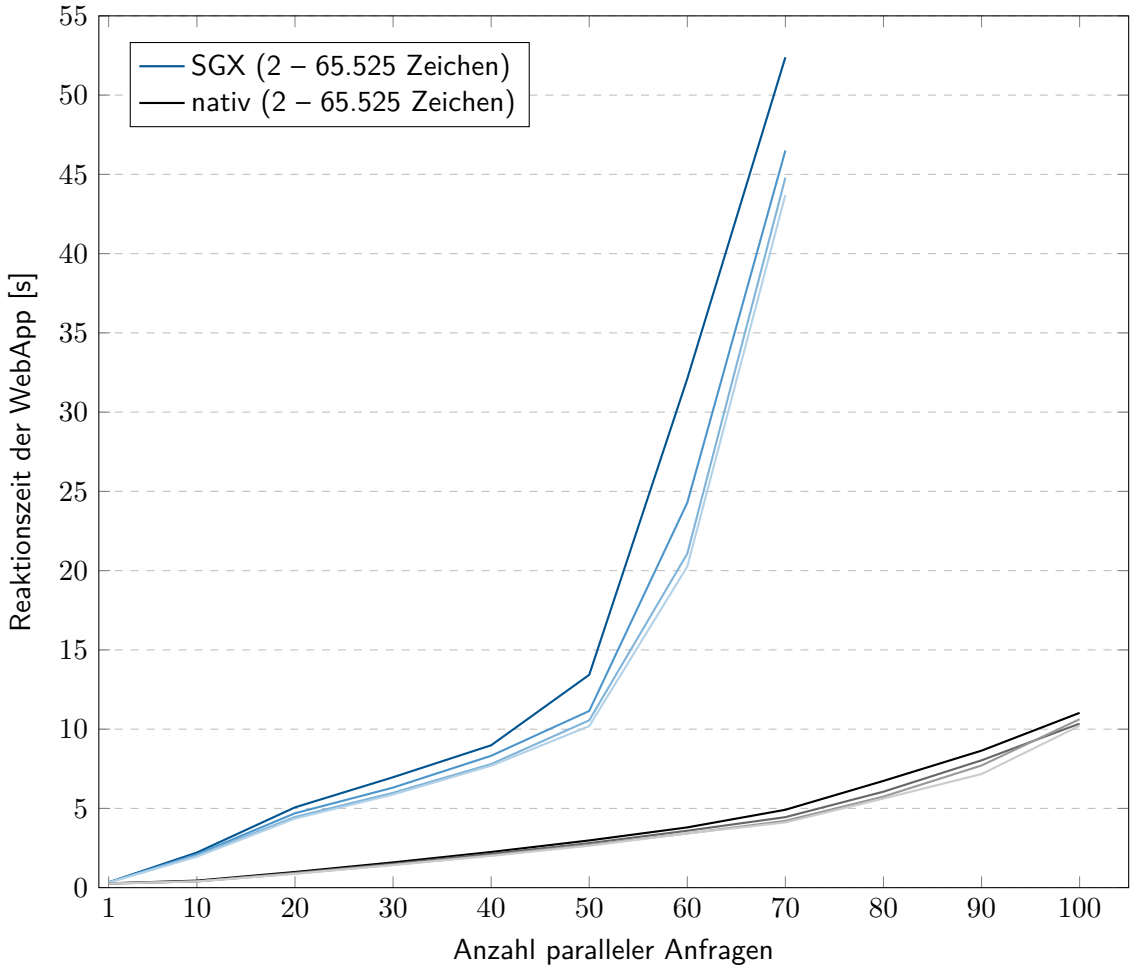


Abbildung 4.5: Einfluss paralleler Zugriffe auf die Reaktionszeit von WordPress bei der Veröffentlichung von Kommentaren

Besonders auffällig ist dabei, dass die Zeit zum Bearbeiten einer Anfrage in letzterem Fall zuerst nahezu linear, ab ca. 50 parallelen Anfragen jedoch viel schneller steigt. Da keiner der drei Dienste bei diesen Testdurchläufen Hinweise wie Fehlermeldungen ausgibt, konnte die Ursache für dieses Verhalten nicht abschließend untersucht werden. Bei der Überwachung der Durchführung dieses Benchmarks konnte allerdings festgestellt werden, dass von den vier zur Bearbeitung von FastCGI-Anfragen bereitstehenden PHP-Prozessen nicht alle über die gesamte Testdauer aktiv sind.
Weiterhin wird deutlich, dass die Länge des Kommentartextes ebenfalls einen Einfluss auf die Reaktionszeit ausübt (so benötigt die WebApp zur Verarbeitung eines 65.525 Zeichen langen Kommentars in der Regel die längste Zeit), dieser allerdings weniger signifikant ausfällt (insbesondere beim nativen Aufbau).
Auch in diesem Testszenario fehlen die letzten Messwerte der SGX-Version von WordPress. Die Reaktionszeiten für 80 parallele Zugriffe und mehr konnten nicht bestimmt werden, da der Anwendungsserver in diesen Fällen die Bearbeitung des entsprechen-

den PHP-Skriptes einstellt. Auch durch mehrere Versuche konnte nicht abschließend geklärt werden, welche Ursache dieses, zu unterschiedlichen Zeitpunkten auftretende, Verhalten aufweist. Weder ein größerer Workerpool von PHP-FastCGI-Prozessen (siehe Listing B.15) noch die Erhöhung des maximalen EPC-Speichers für jede Enklave (siehe Abschnitt 4.2.3.4) ändert etwas an diesem Verhalten.

4.2.3.4 Datei-Upload

In einem letzten Test wurde die Zeit zum Hochladen verschieden großer Dateien über die WordPress-Weboberfläche mittels Wireshark ermittelt.
Wieder kam sowohl die SGX- als auch die native Variante der Webanwendung zum Einsatz, allerdings mit der Einschränkung, dass die TLS-Verschlüsselung des HTTP-Verkehrs in beiden Fällen deaktiviert wurde.
Die Ergebnisse dieser Versuche werden (unter Verwendung einer logarithmischen Skala) in Abbildung 4.6 veranschaulicht.

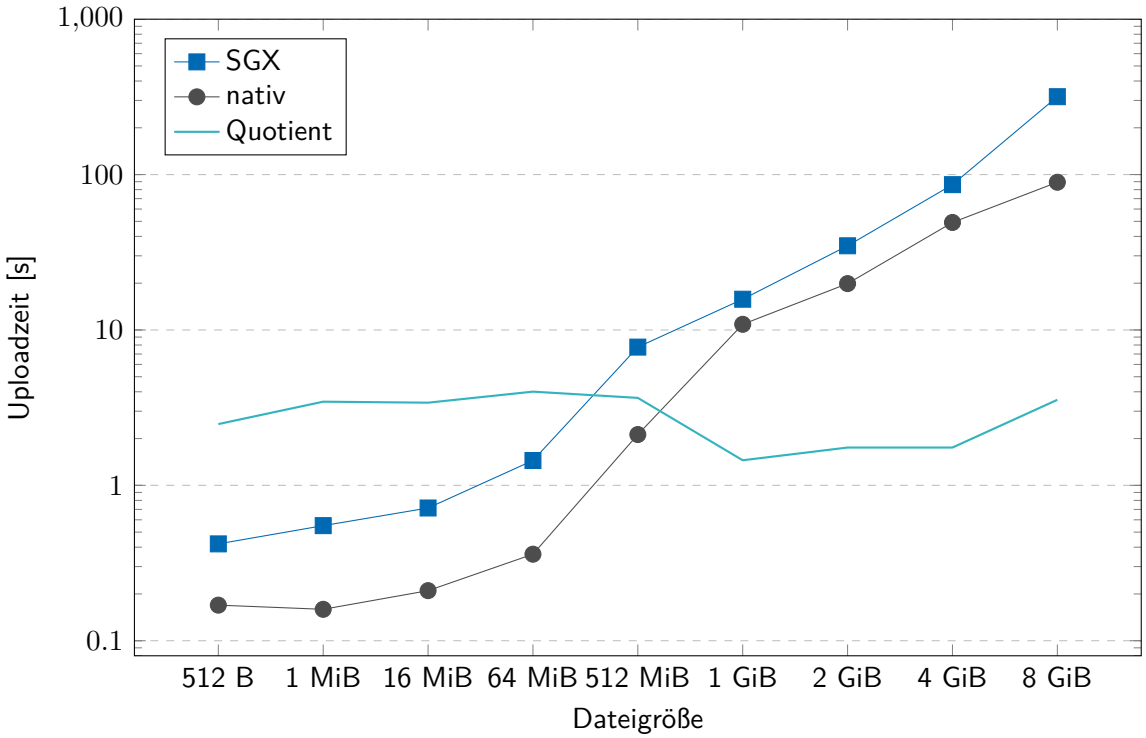


Abbildung 4.6: Einfluss der Dateigröße auf die Upload-Zeit von WordPress

Beim Vergleich der beiden Kurven wird deutlich, dass die native WordPress-Version die Uploads in jedem Fall schneller verarbeiten kann als die in Enklaven ausgeführte. Der Quotient der Uploadzeiten beider Alternativen (und damit der durch SGX verursachte Einfluss auf die Verarbeitungsgeschwindigkeit) variiert zwischen 1,5 und vier, steht jedoch in keinem erkennbaren Zusammenhang mit der Dateigröße.

Im Gegensatz zu den anderen drei Testszenarien konnten in diesem Fall alle Versuche erfolgreich durchgeführt werden. Dabei wurde allerdings festgestellt, dass für das Hochladen von Dateien mit mehr als 1 GiB Inhalt der maximal verfügbare Enklavenspeicher für den Webserver erhöht werden muss. Um alle verwendeten Testdateien zu unterstützen, wird dieser in der *Manifest*-Vorlage `lighttpd.manifest.template` auf 8 GiB gesetzt[23]. Die modifizierte Konfiguration zur erfolgreichen Durchführung der Testdurchläufe mit größeren Dateien wird in Listing 4.8 (Z. 56) aufgezeigt.

```
54 # Konfiguration der SGX-Enklave
55 # max. EPC-Speicher für SGXv1
56 sgx.enclave_size = 8G
57 # max. Anzahl an Enklaven-Threads für SGXv1 (2 für Graphene, ↩
      ↪1 für Lighttpd)
58 sgx.thread_num = 3
```

Listing 4.8: Ausschnitt aus der modifizierten *Manifest*-Vorlage für Lighttpd

4.3 Auswertung

Nachdem die beispielhafte Migration der Webanwendung WordPress durchgeführt und anhand von vier Testszenarien die Unterschiede bezüglich der Performanz bestimmt wurden, sollen die praktischen Konsequenzen dieser Migration im Folgenden diskutiert werden.

4.3.1 Funktionalität und Sicherheit

Durch die Durchführung der Migration von WordPress konnte festgestellt werden, dass die in Kapitel 3 vorgestellte Methodik zur Migration von Multi Service Applications in der Praxis anwendbar ist.
Nach Vollendung der vier Schritte liegt eine Anwendung vor, die allen funktionalen Anforderungen gerecht wird. Zusätzlich werden alle drei Dienste (Webserver, Anwendungsserver und Datenbank) unter Nutzung der Migrationsansätze Graphene-SGX und SCONE innerhalb von SGX-Enklaven ausgeführt und die zwischen ihnen sowie zur Umwelt bestehenden Kommunikationskanäle kryptografisch abgesichert. Eine Ausnahme hierzu stellt der Datenverkehr zwischen dem Webserver Lighttpd und dem

[23] Während die maximale Größe des EPC durch Hardwarevorgaben des verwendeten Prozessors auf 128 MiB festgesetzt ist (siehe Abschnitt 2.1.4.1), kann der Enklaven zur Verfügung gestellte Gesamtspeicher aufgrund des EPC-Pagings dennoch größer sein.

PHP-FPM dar, welcher im Rahmen dieser Arbeit aufgrund fehlender Unterstützung einer Verschlüsselung durch das FastCGI-Protokoll oder den verwendeten Migrationsansatz ungeschützt bleibt.
Mit dem Ziel, die grundsätzliche Durchführbarkeit und Korrektheit der Migrationsmethodik zu überprüfen, wurden außerdem verschiedene weitere Angriffspunkte der Webanwendung nicht abgesichert, obwohl dies im praktischen Einsatz erforderlich ist. Konkret sollten die folgenden Maßnahmen zur Steigerung der Sicherheit vorgenommen werden.

Deaktivierung von HTTP Die unsichere Protokollvariante zur Kommunikation zwischen Nutzer und Webserver sollte durch eine Anpassung der Lighttpd-Konfiguration deaktiviert und HTTPS forciert werden.

Verschlüsselung von FastCGI Die Kommunikation zwischen Web- und Anwendungsserver muss für eine sichere Gesamtanwendung verschlüsselt werden, wobei die Terminierung nur innerhalb der jeweiligen Enklave erfolgen darf. Da FastCGI selbst keine Option zur Verschlüsselung bietet [Bro96], muss diese entweder mit Hilfe externer Werkzeuge integriert oder auf ein alternatives Protokoll zurückgegriffen werden (was beim Anwendungsserver PHP nicht möglich ist).
Eine Option für erstere Alternative stellt die (für beide Kommunikationspartner) transparente Verschlüsselung der FastCGI zugrundeliegenden TCP-Pakete mittels TLS dar[24]. Eine solche Funktion wird beispielsweise durch die Shielding Layer SCONE in Form des mit der „Standard“-Lizenz verfügbaren *Network Shields* bereitgestellt [Arn+16, S. 694 f.]. Das im vorliegenden Beispiel verwendete Library OS Graphene-SGX bietet dagegen keine transparente Verschlüsselung des TCP-Verkehrs an, das Hinzufügen dieser Funktion ist aber aufgrund der Open-Source-Lizenz grundsätzlich möglich.

Attestation Wie bereits in Abschnitt 3.4 erläutert, ist in der Praxis für jede verwendete Enklave die (Remote) Attestation erforderlich, um die Korrektheit der Ausführungsumgebung und des *tCodes* zu überprüfen.
Beide im WordPress-Beispiel verwendeten Migrationsansätze unterstützen diese SGX-Funktion auf unterschiedliche Weise. Während SCONE die Verwendung eines eigenen „Configuration and Attestation Services“ (kurz: CAS) mit erweiterten Funktionen[25] erfordert [SCO20c], integriert Graphene-SGX die für eine

[24] Alternativ können auch andere Herangehensweisen zur kryptografischen Absicherung der FastCGI-Kommunikation umgesetzt werden, das libOS „SGX-LKL“ schützt z.B. jeglichen Netzwerkverkehr durch die Funktionalität eines Virtual Private Networks (kurz: VPN) [Pri+20, S. 2].
[25] Der SCONE CAS bietet zusätzlich zur Local und Remote Attestation verschiedene Funktionen wie automatisches Secret Provisioning oder die Zwischenspeicherung von IAS-Anfragen [SCO20b].

Remote Attestation notwendige Kommunikation in den Verbindungsaufbau und die Zertifikate einer TLS-Sitzung [OSC20a].

Secret Provisioning Jegliche kryptografische Schlüssel und Zertifikate, die innerhalb einer Enklave benötigt werden, dürfen dem zugrundeliegenden Betriebssystem nicht zugänglich sein. Bei der beispielhaften Migration von WordPress wurden solche vertraulichen Informationen mittels Umgebungsvariablen oder ungeschützten Dateien übermittelt. In der Praxis müssen sie jedoch nach erfolgreicher Attestation von einem vertrauenswürdigen Rechner bereitgestellt werden. Zu diesem Zweck können z.B. der bereits erwähnte SCONE CAS oder die Graphene-Werkzeuge für Remote Attestation und Secret Provisioning (Beispiel verfügbar unter [OSC20b]) verwendet werden.
Im Einzelnen ist eine sichere Bereitstellung der folgenden Daten erforderlich[26].

- das für die Datenbank erforderliche SCONE File System Protection File sowie der zugehörige Schlüssel (siehe Abschnitt 4.1.4.1)
- die zur Absicherung der Kommunikation C(Client, Lighttpd) und C(PHP-FPM, MariaDB) verwendeten TLS-Zertifikate und privaten Schlüssel (siehe Abschnitt 4.1.4.1 und 4.1.4.2)
- jegliche für eine sichere FastCGI-Kommunikation verwendeten Schlüssel oder Zertifikate

Dateiverschlüsselung für WS und AS Beim Erstellen der *Manifest*-Vorlage für Lighttpd und PHP-FPM in Graphene-SGX wurden alle veränderlichen Dateien als `allowed_files` spezifiziert. Damit werden deren Inhalte nicht vor unerlaubten Zugriffen geschützt und es muss davon ausgegangen werden, dass vertrauliche Daten sowohl ausgelesen als auch verändert werden können.
Um dies zu verhindern, ist in der Praxis dagegen für alle Dateien mit vertraulichem Inhalt die Deklaration als `protected_files` erforderlich. Somit werden diese durch das Library OS mittels SGX Sealings verschlüsselt. Der dafür verwendete Schlüssel wird nach erfolgreicher Remote Attestation durch Secret Provisioning bereitgestellt [OSC20c; OSC20d].
Insbesondere sind davon die von beiden Diensten verwendeten Logdateien, temporären Verzeichnisse sowie der Ordner mit allen WordPress-Dateien (`<WPDir>`)

[26] Die Zugangsdaten (des *root*- und WordPress-Nutzers) zur Datenbank werden ebenfalls über (unsichere) Umgebungsvariablen bereitgestellt.
Eine Provisionierung ist in diesem Fall jedoch nicht zwingend erforderlich, da die vertraulichen Daten nur während der Initialisierung, nicht aber bei der eigentlichen Ausführung des MariaDB-Servers, benötigt werden. Wird ersterer Schritt sowie die Erstellung des FSPF auf einem vertrauenswürdigen Rechner durchgeführt, müssen ausschließlich die durch das SCONE *File System Shield* verschlüsselten Daten an den Zielrechner übertragen werden.

betroffen. Alle Betriebssystemdateien (z.B. jene unter `/etc/`) bedürfen dagegen keines solchen Schutzes, da sie zur Interaktion mit dem nicht vertrauenswürdigen Teil des Systems dienen und deren Inhalt deshalb in der Enklave grundsätzlich überprüft werden muss.

Kernelpatches für Graphene-SGX Wie in Abschnitt 4.1.3 erläutert, benötigt dieses Library OS spezielle Systemfunktionen, die entweder durch bestimmte Kernelpatches oder durch einen Graphene-spezifischen Treiber bereitgestellt werden [Gra19a]. Laut der Dokumentation des Letzteren weist dieser jedoch Sicherheitslücken auf, die im produktiven Einsatz vermieden werden sollten, weshalb in diesem Fall nur die erste Option als valide angesehen wird [OSC20f].

4.3.2 Performanz

Nach Durchführung der Last- und Geschwindigkeitstests anhand vierer Szenarien und der grafischen Veranschaulichung der Ergebnisse wird deutlich, dass die Verwendung von sicheren Enklaven die Performanz einer Multi Service Application merklich beeinflusst.
Die geringste Geschwindigkeitseinbuße kann beim Download von Dateien beobachtet werden, da in diesem Fall nur ein einzelner Dienst der A_{MS} in die Anfrage involviert ist und nur dieser die durch SGX verursachte Verzögerung mit sich bringt. In allen anderen Fällen sind sowohl WS und AS als auch die DB an der Bearbeitung einer Nutzeranfrage beteiligt und die Reaktionszeit wird durch jeden einzelnen erhöht.
Dass auch im vierten Szenario die Verringerung der Performanz relativ gering ausfallen kann, ist damit zu erklären, dass beim Hochladen einer Datei DB- und Anwendungsserver nur wenige Berechnungen durchführen müssen und deren Umfang nicht von der Dateigröße abhängt. Der am stärksten involvierte Dienst ist auch in diesem Fall der WS.
Zusammengefasst wird die Performanz der Webanwendung um die in Tabelle 4.2 aufgeführten (gerundeten) Faktoren reduziert (bzw. die benötigte Zeit erhöht).

Generell kann diese Verringerung durch unterschiedliche Bestandteile der $A_{MS,p}$ verursacht werden. Wie in Abschnitt 2.1.7 erläutert, besitzt Intel SGX selbst aufgrund seiner Sicherheitseigenschaften einen großen Einfluss auf die für Rechenoperationen in Enklaven benötigte Zeit.
Zusätzlich bedingen aber auch die jeweils verwendeten Migrationsansätze (in diesem Fall SCONE und Graphene-SGX) die Performanz der ausgeführten Applikation. So zeigen z.B. Tian et al. [Tia+17] bzw. Shen et al. [She+20] Unterschiede zwischen mehreren Library-OS-Implementierungen bezüglich der Leistungsfähigkeit portierter

	Szenario	Beschreibung	beteiligte Dienste	relative Reaktionszeit
Lesen	1	Aufrufen eines veröffentlichten Blogposts	WS, AS, DB	4x
	2	Herunterladen einer von WordPress bereitgestellten Datei	WS	2x
Schreiben	3	Veröffentlichen eines Kommentars zu einem vorhandenen Post	WS, AS, DB	4-10x
	4	Hochladen einer Datei	WS, AS, DB	1,5-4x

Tabelle 4.2: Performanzeinbußen von WordPress in SGX

Anwendungen auf.
Zuletzt besitzt auch die vorgestellte Migrationsmethodik bzw. die in den einzelnen Schritten getroffenen Entscheidungen einen eigenen Einfluss. Wie bereits ausgeführt, steht beispielsweise die Größe der TCB in engem Zusammenhang mit der Geschwindigkeit und Sicherheit einer SGX-Anwendung, weshalb eine u.U. vorgenommene Partitionierung der in SGX auszuführenden Anwendung maßgeblich relevant für die gesamte Verringerung der Reaktionszeit ist.

Die durch die Software Guard Extensions verursachte Reduktion in der Verarbeitungsgeschwindigkeit kann mit dem zusätzlichen Rechenaufwand zur Ver- und Entschlüsselung sowie zur Verwaltung des EPC erklärt werden.
Da dessen Speicher für drei komplett in Enklaven ausgeführte Dienste nicht ausreicht[27], müssen durch den Intel-SGX-Treiber mehrere EPC-Seiten in den Systemspeicher ausgelagert werden (siehe Abschnitt 2.1.4.1). Der dafür zuständige Hintergrunddienst wird `ksgxswapd` genannt und beansprucht während der Performanztests bis zu 50% der CPU-Leistung.
Zusätzlich erfordern die in Graphene-SGX ausgeführten Dienste (WS und AS) für jeden Systemcall einen Enclave Exit und nach dem Funktionsaufruf einen erneuten Wechsel in den Enclave Mode, was ebenfalls einen großen Einfluss auf die Verarbeitungsgeschwindigkeit bedeutet. Die Auswirkung dieser Ursache könnte allerdings durch eine Modifikation der entsprechenden *Manifest*-Dateien reduziert werden, da das libOS die Möglichkeit zur Verwendung von asynchronen „Switchless Calls“ für

[27] Für die native Ausführung von WS, AS und DB wurde (ohne aktive Nutzeranfragen) mittels des Linux-Werkzeugs `pmap` [pro20] ein gesamter Speicherbedarf von mehr als einem Gibibyte ermittelt.

Systemaufrufe nach [Tia+18] bietet [Gra19e].
Eine weitere Verzögerung bei allen Anfragen, in die der Datenbankserver involviert ist, wird durch die Verwendung des SCONE File System Protection Files verursacht. Dieser verschlüsselte Zugriff auf jegliche Dateien der Datenbasis erfordert zusätzlichen Rechenaufwand – jeweils nach dem Laden des Dateiinhaltes in die Enklave und vor dem Übertragen vom DB-Server zurück in den $\mathcal{P}_u$. Da es sich bei dieser Sicherheitsmaßnahme um eine SCONE-Funktionalität handelt, kann das Vergleichsszenario ohne SGX-Einsatz in dieser Hinsicht nicht an den sicheren WordPress-Aufbau angeglichen werden.

Weiterhin kann festgestellt werden, dass in drei der vier getesteten Szenarien bei Verwendung der SGX-Installation von WordPress eine oder mehrere Messungen nicht durchgeführt werden konnten, weil einzelne Dienste nicht allen Anforderungen (bezüglich der Parallelität oder Dateigröße) gerecht werden konnten. Da die Ursachen dafür sehr vielfältig sein können, war es im Rahmen dieser Masterarbeit nicht in jedem Fall möglich, abschließend zu klären, wodurch bei bestimmten Lastniveaus ein Dienstabsturz oder die Beendigung des Verarbeitungsvorgangs hervorgerufen wurde. Eine weitere Untersuchung kann deshalb bei der Identifikation der Auslöser und Probleme sowie bei deren Behebung helfen.
Insbesondere der in Graphene-SGX ausgeführte Webserver konnte in zwei von drei Fällen als Schwachpunkt ausgemacht werden. Möglicherweise kann die Verwendung einer anderen Serverimplementierung dabei zur Unterstützung größerer Downloads führen (Anwendungsfall zwei). Eine erfolgreiche Verarbeitung von mehr als 80 parallelen Zugriffen im ersten Testszenario könnte dagegen durch Anpassung des Speicherlimits im Graphene-Quellcode oder durch Anwendung eines anderen Migrationsansatzes (bzw. Library OS) erreicht werden.
Ist eine sehr hohe Leistungsfähigkeit der gesamten Multi Service Application – und insbesondere eines einzelnen Dienstes – vonnöten, kann auch erwogen werden, jenen Anwendungsteil mit den höchsten Performanzanforderungen auf vertrauenswürdigen Rechnern auszuführen und somit die Notwendigkeit für sichere Enklaven zu eliminieren.

4.3.3 Zusammenfassung

Nach der Durchführung und Auswertung der beispielhaften Migration von WordPress kann festgehalten werden, dass die zusätzliche Sicherheit, welche der Einsatz von Intel SGX bietet, untrennbar mit der Inkaufnahme einer verringerten Performanz für die ausgeführte Anwendung verbunden ist. Je nach Anwendungsszenario muss dabei

individuell entschieden werden, wie viel Aufwand bei der Migration vertretbar ist sowie welche Anforderungen an die Sicherheit und Leistungsfähigkeit der portierten (serviceorientierten) Anwendung gestellt werden.
Im beispielhaften Fall von WordPress wurde mit vergleichsweise geringem Aufwand die komplette Webanwendung (mit den erwähnten Ausnahmen) durch die Verwendung von sicheren Enklaven und kryptografischen Verfahren vor Angriffen auf die Integrität oder Vertraulichkeit der verarbeiteten Daten geschützt.
Aufgrund des (zeitlichen) Umfangs dieser Arbeit stehen noch diverse Angriffspunkte aus, die für einen praktischen Einsatz eliminiert werden müssen. Wie beschrieben, ist dies in den meisten Fällen jedoch unter Zuhilfenahme der existierenden Werkzeuge und Hilfsmittel (ohne die Notwendigkeit für Neuentwicklungen jeglicher Art) möglich. In Bezug auf die Performanz müssen verschiedene Einschränkungen akzeptiert werden (z.B. bei der maximalen Anzahl paralleler Verbindungen), die für den Gebrauch als Veröffentlichungswerkzeug für private Blogs als unproblematisch eingeschätzt werden, für umfangreichere Einsatzszenarien jedoch ebenfalls einer Mitigation bedürfen.

5 Migration von Container-Workloads

Bereits bei der praktischen Evaluation der in Kapitel 3 vorgestellten Methodik zur Migration von Multi Service Applications wurden (Docker-) Container verwendet (siehe Abschnitt 4.1.4.1). Unabhängig davon kommt die zugrundeliegende Containervirtualisierung in der Praxis für die verschiedensten Anwendungsszenarien zur Softwareentwicklung und -betrieb immer häufiger zum Einsatz (insbesondere für Multi Service Applications). Gerade im Umfeld von agilem Projektmanagement und DevOps[1] wird die Möglichkeit zur Bereitstellung von Anwendungen bzw. Diensten in Containern verwendet, da sie mit diversen Vorteilen verbunden ist.
So wird z.B. eine gewisse Isolation von Prozessen und Abstrahierung von Ressourcen, ähnlich wie bei der Verwendung virtueller Maschinen, ermöglicht. Gleichzeitig weisen Container aber einen schnelleren Startvorgang und geringeren Ressourcenanspruch als Letztere auf[2] [CMD16, S. 55].
In diesem Kapitel soll deshalb diskutiert werden, inwiefern die Sicherheit von SGX-Enklaven durch in Containern ausgeführte (sog. „containerisierte“) Software genutzt werden kann. Dabei wird aufgrund ihrer weiten Verbreitung[3] im Folgenden Bezug auf die Containerplattform „Docker“ genommen. Eine Adaption auf alternative Umgebungen ist aufgrund eines ähnlichen Funktionsprinzips jedoch grundsätzlich möglich [CMD16, S. 55 f.].
Abschließend wird zusätzlich die Möglichkeit der Verwaltung solcher SGX-Container durch Orchestrierungsplattformen betrachtet.

5.1 Grundlagen der Containervirtualisierung

Um den Unterschied zwischen Containern und herkömmlichen Anwendungen zu verstehen, wird im Folgenden die grundlegende Funktionsweise Ersterer erläutert.
Dabei wird die Ausführung von Docker auf einem Linux-Betriebssystem vorausge-

[1] Der Begriff „DevOps“ setzt sich aus den zwei Teilen *Development* und *Operations* zusammen. Das Ziel dieses Ansatzes ist eine starke und vereinfachte Zusammenarbeit zwischen diesen beiden Abteilungen eines Unternehmens zur Steigerung der Flexibilität und Effizienz [Kör12, S. 3 ff.].
[2] Felter et al. stellten beispielsweise einen um 38% höheren Performance-Overhead von virtuellen Maschinen im Vergleich zu Containern fest [Fel+15, S. 171].
[3] Nach dem amerikanischen Unternehmen für Cloud-Sicherheit „Sysdig“ basierte 2019 ca. 79% der weltweiten Containernutzung auf Docker [Car19].

setzt, da die Containerplattform ursprünglich für dieses entwickelt wurde. Seit 2016 wird Docker zwar auch auf Windows-Systemen unterstützt [Doc20e], intern wird für die Containerverwaltung jedoch auch in diesem Fall auf Linux-Funktionalität zurückgegriffen [Doc20g].

Das Konzept eines „Containers“ basiert auf einer abgeschirmten Umgebung auf dem Hostrechner, in der ein oder mehrere Prozesse zur Ausführung kommen. Dabei werden alle Zugriffe auf „interne“ Ressourcen (z.B. Dateien oder andere Prozesse innerhalb dieses Containers) ermöglicht, jene auf „externe“ Ressourcen (die des Hostsystems) jedoch unterbunden.
Zur Erstellung, Verwaltung und Interaktion mit allen, auf einem Rechner ausgeführten Containern wird ein mit hohen Berechtigungen ausgeführter Systemdienst (der „Docker Daemon“) verwendet [CMD16, S. 56 f.].

Um die beschriebene Isolation zu erreichen, werden verschiedene Features des Linux-Kernels genutzt, allen voran *namespaces*, *cgroups* und *capabilities*. Erstere erlauben die Separierung der Sicht auf Systemressourcen in verschiedene Namensräume, wodurch mehrere Prozesse (bzw. -gruppen) z.B. alle verfügbaren Netzwerkschnittstellen oder Prozess-IDs als ihnen allein zugehörig wahrnehmen [CMD16, S. 56]. Ergänzend werden *cgroups* (control groups) verwendet, um die Ressourcennutzung eigenständiger sowie untergeordneter Prozesse einzuschränken und damit die maximale, durch diese verursachte, Ressourcenauslastung zu begrenzen [KM19, S. 256]. Durch die Verwendung von *capabilities* wird schließlich ein feingranulares Rechtemanagement der ausgeführten Prozesse ermöglicht [HM08, S. 164].
Weiterhin stellt der Docker Daemon jedem Container sein eigenes, virtuelles Dateisystem mittels *union mountings* zur Verfügung, welches prinzipiell nur flüchtigen Speicher umfasst. Da dessen Inhalt nach jedem Containerneustart zurückgesetzt wird, ist nur durch das mit einer verminderten Isolation einhergehende Einbinden von Host-Verzeichnissen eine persistente Datenspeicherung möglich [TRA15, S. 73].
Im Gegensatz zu virtuellen Maschinen werden verschiedene OS-Ressourcen wie der Kernel und – wenn möglich – *shared libraries* zwischen mehreren Containern sowie mit dem Host geteilt [CMD16, S. 55].

Zusammengefasst beinhaltet also ein „Container“ als Einheit eine Menge von Prozessen mit deren *namespace*, erteilten *capabilities* und virtuellen Dateisystem, während seine Ressourcenzugriffe durch entsprechende *cgroups* kontrolliert werden. Gleichzeitig interagiert er mit der Umwelt durch die Nutzung eines gemeinsamen Kernels und geteilter Bibliotheken.

5.2 Übertragung des Migrationsprinzips auf Container

In Containern befindliche Anwendungen (kurz: A_{Cont}) unterscheiden sich demnach von ihren direkt im Host-OS ausgeführten Pendants hauptsächlich durch die zusätzliche Abstraktionsschicht und eingeschränkte Ressourcennutzung. So ist für den Zugriff auf Schnittstellen wie jene zum Dateisystem oder Netzwerk stets der Umweg über den Docker Daemon erforderlich, welcher Berechtigungsprüfungen und die Übersetzung zwischen logischen und physischen Ressourcen durchführt. Die Ähnlichkeit der beiden Alternativen wird dadurch deutlich, dass eine bereits entwickelte Anwendung in der Regel mit wenig Aufwand als Container-Image bereitgestellt und somit in einem solchen ausgeführt werden kann (siehe z.B. [Toz16]).
Für die SGX-Migration einer A_L bedeutet dies, dass die grundlegende Vorgehensweise – mit wenigen Änderungen – auch angewendet werden kann, wenn sie sich in einem Container befindet. Solange die zur Nutzung der sicheren Umgebung erforderlichen Schnittstellen und Ressourcen innerhalb dessen zur Verfügung stehen, kann auch eine A_{Cont} (komplett oder teilweise) in einer Secure Enclave ausgeführt werden. Im Beispiel der Migration von WordPress wurde dies anhand des DB-Servers bereits gezeigt, welcher mittels SCONE innerhalb eines Docker-Containers gestartet und durch eine SGX-Enklave geschützt wird.

Grundsätzlich bestehen zwei Möglichkeiten zur geschachtelten Nutzung von Containern und Intel SGX.
So ist das Starten einer Anwendung in einem Container denkbar, welcher sich seinerseits innerhalb einer Enklave befindet. Da auf diese Weise Letztere alle Isolations- und Virtualisierungsmechanismen für den Container beinhalten muss, ist entweder eine umfassende Schnittstelle für die (verwaltende) Kommunikation mit dem Docker Daemon oder dessen Integration in die gleiche Enklave erforderlich.
Alternativ ist die Ausführung der Anwendung in einer Secure Enclave möglich, welche ihrerseits Teil eines Containers ist. Bei dieser Möglichkeit kann sich jegliche Funktionalität zu dessen Verwaltung außerhalb der Enklave befinden, analog zum herkömmlichen Einsatz von Docker abseits des SGX-Umfelds. Dadurch wird eine deutlich reduzierte TCB im Vergleich zur ersten Alternative erreicht, sowie die Komplexität reduziert. Hinsichtlich der Sicherheit weist die Positionierung der Containerverwaltung im nicht vertrauenswürdigen Teil des Systems keine Einschränkungen auf, da die schützenswerten Daten ausschließlich von der A_{SGX}, nicht aber vom Docker-Umfeld benötigt werden. Folglich müssen sich diese Daten nie außerhalb der Enklave befinden.
Ausgehend von den beschriebenen Argumenten, wird an dieser Stelle eine Entscheidung

für die letztgenannte Alternative getroffen, welche die in Abbildung 5.1 dargestellte Struktur aufweist. Dabei ist zur Verwendung von nicht für SGX entwickelten Anwendungen außer der eigentlichen Enklave das Vorhandensein der eingesetzten Migrationswerkzeuge sowie aller benötigter Dateien und Informationen erforderlich.

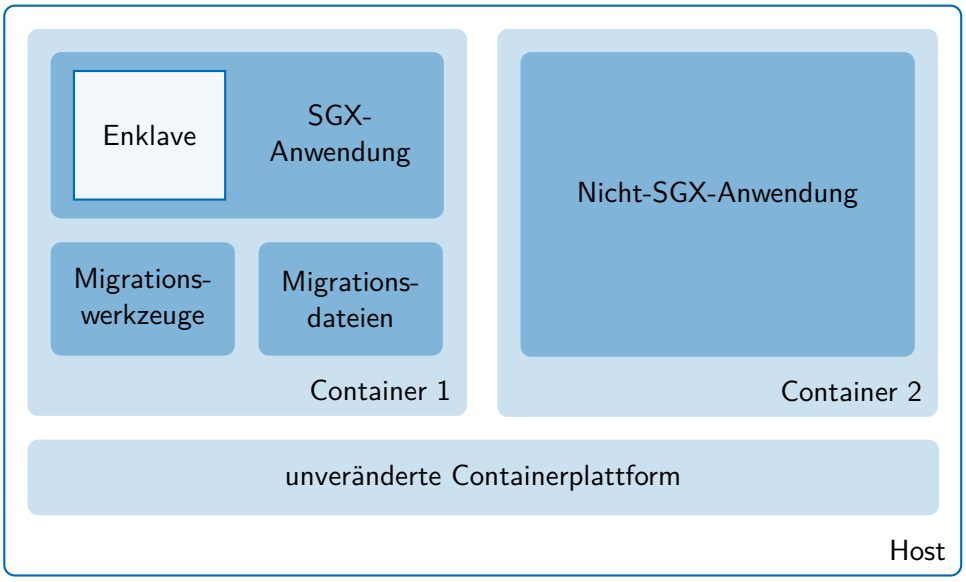


Abbildung 5.1: Aufbau einer containerisierten SGX-Anwendung

Im Folgenden wird der Einfluss der Containerumgebung auf die Vorgehensweise bei der Migration von Legacy Applications in das SGX-Umfeld aufgezeigt. Dabei sollen sowohl aus einem einzelnen Dienst bestehende als auch serviceorientierte Anwendungen berücksichtigt werden.

5.2.1 Migration von einfachen Anwendungen

Liegt die A_L in Form eines Docker-Images vor, sind zusätzlich zu den bei der Migration einer herkömmlichen Anwendung zu befolgenden Schritten weitere Vorbereitungen erforderlich.
Da nicht der gesamte Container, sondern die darin enthaltene Anwendung migriert wird, muss zuerst das ausgeführte Programm bestimmt werden, wobei nach dem im Containerumfeld vermehrt umgesetzten „Microservice“-Prinzip meist jeweils eine einzelne Anwendung enthalten ist. Dazu kann eine Analyse der diesem Containerabbild zugrundeliegenden Konfigurationsdatei `Dockerfile` erfolgen, dessen Syntax z.B. in [Doc20f] erläutert wird. Die durch den Daemon zu startende Programmdatei wird in der Regel durch einen darin enthaltenen `ENTRYPOINT`- oder `CMD`-Befehl (bzw. in einem dabei angegebenen Shellskript) spezifiziert.
Im Fall des offiziellen `Dockerfiles` der In-Memory-Datenbank „Redis“ in der Version

6.0 wird beispielsweise die Binärdatei `redis-server` zum Starten der A_{Cont} verwendet [Doc20b].

Nachdem die Programmdatei identifiziert wurde, kann eine Portierung unter Verwendung eines vorhandenen Migrationsansatzes erfolgen, wobei diese gänzlich innerhalb des Containers durchgeführt wird und dessen Gegebenheiten statt denen des Hostsystems berücksichtigt werden müssen. Da sich z.B. die verwendete Linux-Distribution bei gleichem Kernel von der des physischen Rechners unterscheiden kann, nutzt die containerisierte Anwendung u.U. abweichende Bibliotheken oder Implementierungen von OS-Diensten.
Wie bereits erwähnt, ist zum späteren Start der Applikation die Verfügbarkeit der SGX-spezifischen Schnittstellen und Ressourcen innerhalb des Containers erforderlich. Aus diesem Grund werden z.B. in Abschnitt 4.1.4.1 beim Start des migrierten MariaDB-Servers (je nach Attestation-Schema unterschiedliche) Dateien des `/dev/`-Verzeichnisses in den Container eingebunden, um auf den SGX-Treiber zugreifen zu können.

5.2.2 Migration von Multi Service Applications

Mit Hilfe von Containern bereitgestellte Multi Service Applications (kurz: $A_{MS,Cont}$) können auf zwei verschiedene Weisen aufgebaut sein.
Die für das serviceorientierte Umfeld typische Vorgehensweise ist die Kapselung jedes einzelnen Dienstes in einem eigenen Container und die Definition von Kommunikationsschnittstellen zur wechselseitigen Interaktion. Zum strukturierten Starten solcher, multiple Container umfassender, Applikationen, kommt z.B. das Werkzeug „Docker Compose“ und eine entsprechende Konfigurationsdatei zum Einsatz [Doc20h].
Trotz der Vorteile dieser unabhängigen Dienstbereitstellung existieren (aus unterschiedlichen Gründen) diverse $A_{MS,Cont}$, welche den gleichen Container für mehrere Dienste einsetzen. In diesem Fall muss eine Separierung der Teilanwendungen durchgeführt werden, da die meisten vorgestellten Migrationsansätze von einer A_L ausgehen, welche aus einer einzelnen Programmdatei und deren Abhängigkeiten besteht. Die gleichzeitige Portierung mehrerer Anwendungen wird dabei nicht unterstützt. Nachdem alle Dienste unabhängig voneinander und sequentiell migriert wurden, können sie entweder wieder in einem gemeinsamen oder jeweils in einem eigenen Container verwendet werden.
In jedem Fall kann eine containerisierte A_{MS} – nach eventueller vorbereitender Separierung – durch Anwendung der in Kapitel 3 vorgestellten Methodik in das SGX-Umfeld migriert werden, ähnlich wie eine gewöhnliche Multi Service Application. Die

in diesem Abschnitt beschriebenen Spezifika von Containeranwendungen (u.a. die Durchführung der Migration innerhalb des Containers) müssen allerdings Beachtung finden.

5.2.3 Unterstützung durch Migrationsansätze

Wie bereits in dem in Anhang A aufgeführten Vergleich der wissenschaftlichen Studien zur Migration bestehender Anwendungen in Intel-SGX-Enklaven erwähnt wurde, bieten einige der aufgeführten Ansätze bereits (verschieden umfangreiche) Unterstützung für den Einsatz von Containern.
So wurden z.B. das Partitionierungsframework „lxcsgx“ sowie die Shielding Layer „SCONE“ explizit mit dem Ziel des Schutzes containerisierter Anwendungen mittels Secure Enclaves entwickelt. Bei den Library-OS-Implementierungen „Graphene-SGX“ und „Occlum“ kam die Möglichkeit zur Migration bzw. Ausführung innerhalb eines Containers dagegen erst nach Veröffentlichung der Studie aufgrund des Vorliegens als Open-Source-Software hinzu. Allen gemein ist jedoch die Konzentration auf eine einzelne Containerplattform (Linux Containers, kurz: LXC, im Fall von lxcsgx bzw. Docker bei den anderen Ansätzen).
Da SCONE und Graphene-SGX bereits bei der Beispielmigration in Kapitel 4 eingesetzt wurden, sollen im Folgenden deren Möglichkeiten hinsichtlich der Portierung einer A_{Cont} zusammengefasst werden.

SCONE Die Shielding Layer setzt die Verwendung von Docker-Containern als Grundlage einer Anwendungsmigration voraus. Jegliche SCONE-Software wird in Form von solchen Container-Images bereitgestellt. Folglich erfolgt auch das für die Migration notwendige Rekompilieren des A_L-Quelltextes ausschließlich innerhalb eines Containers [SCO20a].
Liegt die zu migrierende Anwendung bereits als Docker-Image vor und basiert dieses auf der Distribution Alpine Linux, bietet der Entwickler eine Automatisierung der Migration als sog. „Sconifying“-Funktionalität an. Sie erlaubt außerdem die teilautomatisierte Konfiguration des SCONE *File System Shields* und wird (für die „Community-Lizenz“) unter [SCO20h] erläutert.
Um eine vorbereitete Applikation durch die SCONE-Laufzeitumgebung in einer Enklave zu starten, ist die Verfügbarkeit des SGX-Treibers im Container erforderlich. Dazu wird bei dessen Start durch das Kommando `docker run` entweder das Blockdevice `/dev/isgx` (bei Verwendung der IAS-Attestation) oder der Ordner `/dev/sgx` (für DCAP) eingebunden (siehe auch Abschnitt 4.1.4.1). Eine Schnittstelle zu den Architectural Enclaves ist dagegen nicht notwendig, da SCONE für deren Funktionalität

seine eigenen Implementierungen bereitstellt.

Graphene-SGX Dieses Library OS unterstützt seit Juli 2020[4] die Migration einer A_{Cont}. Das genutzte Werkzeug wird „Graphene Shielded Containers“ (kurz: GSC) genannt und erlaubt die libOS-Ausführung – und damit den Start der enthaltenen Anwendung – innerhalb eines Docker-Containers. Außerdem besteht die Möglichkeit, vorhandene, auf Ubuntu (Version 18.04) basierende, Container (teil-) automatisiert zu migrieren, wobei die benötigten *Manifest*-Dateien von GSC generiert werden [Gra19b]. Um die Schnittstellen zur SGX-Umgebung innerhalb des Containers verfügbar zu machen, ist ergänzend zu dem bei SCONE verwendeten Parameter das Einbinden des Graphene-spezifischen Treibers `/dev/gsgx` sowie des Sockets zum AESM erforderlich. Wird von dem Docker-Image `<Image-Name>` ausgegangen, kann dessen GSC-Version z.B. durch das folgende Kommando gestartet werden.

```
1 docker run \
2   --device=/dev/isgx
3   --device=/dev/gsgx
4   -v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket
5     "gsc-<Image-Name>"
```

Listing 5.1: Starten der GSC-Version eines Containers

5.2.4 Beispielszenario

Zur Veranschaulichung der beschriebenen Besonderheiten bei der Migration von containerbasierten Multi Service Applications soll wieder das CMS WordPress zum Einsatz kommen, welches durch die Verwendung von Docker-Containern installiert werden kann.
Durch die im öffentlichen Image-Repository „Docker Hub“ verfügbare Anleitung wird aufgezeigt, dass zum Betrieb eines vollwertigen WordPress-Systems zwei Container mit den Images `wordpress` resp. `mysql` notwendig sind [Doc20i]. Durch Analyse der jeweiligen `Dockerfiles` sollen die gestarteten Anwendungen identifiziert werden. Während beim Datenbankserver nur einen einzelnen Dienst enthalten ist, wird deutlich, dass das `wordpress`-Image zwei Anwendungen beinhaltet; den Anwendungsserver PHP sowie den Apache-Webserver httpd [Doc20d].
Um diese zwei Dienste vor der Migration voneinander zu trennen, könnte beispielsweise auf einen alternativen Webserver ausgewichen werden, welcher unabhängig von PHP

[4] Die Graphene Shielded Containers wurden mit der folgenden Commit-ID im offiziellen GitHub-Repository eingeführt: *f51aafddf3989db89fd4375c95430d9ab4d0fe29* [OSC20e].

gestartet wird. In diesem spezifischen Fall ist jedoch auch die gemeinsame Migration beider Dienste möglich, da es sich aufgrund der Integration des AS in den WS als dynamische Bibliothek trotzdem um eine einzelne Anwendung (und Programmdatei) handelt (siehe Abschnitt 4.1.1, Fußnote 4).

Um alle in den verwendeten Containern enthaltenen Anwendungen zur Verwendung von Intel SGX zu migrieren, kann die in Kapitel 3 beschriebene Methodik genutzt werden. Bei der Auswahl zu verwendender Migrationsansätze sollten für einen minimalen Aufwand jedoch jene bevorzugt werden, die eine Unterstützung für Docker-Container mitbringen.
In diesem Beispiel könnte aufgrund der Verfügbarkeit des bereits einsatzbereiten, portierten Images wieder SCONE für den DB-Server zum Einsatz kommen. Da der zweite Container auf dem Image `php:7.4-apache`, und dieses wiederum auf der Ubuntu ähnlichen Linux-Distribution „Debian“, basiert [Doc20a], kann möglicherweise mit Hilfe von GSC eine teilautomatisierte Migration vorgenommen werden. Ist dies nicht möglich, kann z.B. auf die Verwendung des eigenständigen (auf Alpine Linux basierenden) PHP-FPM-Containers in Kombination mit einem beliebigen Webserver (wie dem ebenfalls auf Alpine-Basis verfügbaren httpd) ausgewichen werden. In diesem Fall wird eine teilautomatisierte Migration beider Container z.B. mittels „Sconifying“ durchgeführt.

5.3 Container-Orchestrierung

Wie bereits beschrieben, können (Docker-) Container durch den Docker Daemon auf einem einzelnen Host gestartet werden. Für den produktiven Einsatz kommerzieller Software ist jedoch häufig eine hohe Verfügbarkeit und möglichst dynamische Skalierbarkeit der ausgeführten Container erforderlich. Um diese Ziele zu erreichen, findet der Betrieb solcher Anwendungen nicht auf einem einzelnen, sondern über mehrere (virtuelle) Rechner verteilt statt, häufig unter Nutzung von (Public-) Cloud-Umgebungen. Die einzelnen Container werden dabei in sog. „Clustern“ zusammengefasst und organisiert.
Zur Vereinfachung und Automatisierung einer solchen verteilten Containernutzung wurden verschiedene Werkzeuge zur Clusterverwaltung („Orchestrierung“) entworfen, z.B. das weit verbreitete „Kubernetes“[5] (weitere Informationen über dessen Nutzung sowie die Containerverwaltung auf verteilten Systemen im Allgemeinen können z.B. [HBB18] entnommen werden). Da eine solche Orchestrierungsplattform die Verwaltung

[5] Im Jahr 2019 machte Kubernetes ca. 77% aller weltweit eingesetzten Orchestrierungsplattformen aus [Car19].

zusammengehöriger Container unterstützt, wird insbesondere die Ausführung von Multi Service Applications stark vereinfacht.
Ein durch Kubernetes organisiertes Cluster beinhaltet immer einen oder mehrere Rechner, auf denen eine Containerplattform sowie ein Kubernetes-Dienst läuft und welche als „Knoten“ des Clusters bezeichnet werden. Die Verwaltung erfolgt dabei zentral von einer „Control Plane“ aus [The20d].
Ausgehend von der bereits diskutierten Nutzung von Intel SGX in Kombination mit Containeranwendungen, sollen im Folgenden Besonderheiten bei der Orchestrierung solcher Container diskutiert werden. Dabei wird – aufgrund der breiten Nutzung – exemplarisch die Plattform Kubernetes herangezogen.

Orchestrierung von Containern mit Secure Enclaves Unter der Voraussetzung, dass zum Schutz von containerisierten Anwendungen durch SGX die in Abbildung 5.1 dargestellte Architektur umgesetzt wird, ist die Nutzung von Orchestrierungsplattformen ohne Einschränkungen möglich. Da die Verwaltung und Steuerung der Container in diesem Fall nicht verändert wird, kann Kubernetes die gleichen Schnittstellen nutzen, die bei einer herkömmlichen A_{Cont} bestehen.
Um die Funktionalität von sicheren Enklaven zu nutzen, müssen auf allen verwendeten Knoten jeweils die gleichen Voraussetzungen wie bei der Ausführung einer A_{Cont} auf einem einzelnen Host gegeben sein. Neben dem Vorhandensein einer entsprechenden CPU ist dafür die Installation der benötigten SGX-Umgebung (z.B. Treiber, siehe Abschnitt 2.1.5) erforderlich. Ein Container, welcher eine (nach Abschnitt 5.2 portierte oder native) SGX-Anwendung enthält, kann schließlich auf einem Kubernetes-Knoten auf gleiche Weise ausgeführt werden, wie auf einem lokalen Rechner – so ist z.B. das Einbinden der Treiberschnittstelle notwendig.
Da die SGX-Spezifika ausschließlich die Clusterknoten sowie die eingesetzten Container betreffen, kann das Orchestrierungswerkzeug beide auf gleiche Weise steuern wie ihre Pendants ohne Enklavennutzung. Eine Unterstützung ist allerdings bei der Zuweisung von Containern zu Hosts möglich, welche grundsätzlich durch Kubernetes automatisiert wird, durch die Spezifikation von Regeln jedoch durch den Nutzer beeinflusst werden kann. Mittels *Labels* bzw. *Annotationen* ist z.B. die Zuordnung diverser Eigenschaften – wie der Unterstützung von SGX – zu einem Knoten möglich [HBB18, S. 65 ff.]. Beim sog. „Deployment“ (siehe [HBB18, S. 133 ff.]) einer A_{Cont} kann anschließend festgelegt werden, dass ein Container mit einer SGX-Anwendung ausschließlich auf solchen Knoten gestartet werden darf, welche die entsprechende Kennzeichnung besitzen (siehe z.B. [The20c]).
Außerdem wäre eine Weiterentwicklung des Orchestrierungswerkzeugs denkbar, sodass beim Deployment für jeden Container einzeln angegeben werden kann, ob dieser SGX

verwendet und der Kubernetes-Dienst alle notwendigen Konfigurationen automatisch vornimmt.
Eine weitere Möglichkeit zur verstärkten Integration der Enklaven-Funktionalität in die Containerverwaltung ist z.B. die Entwicklung von Kubernetes-Plugins, welche verschiedene SGX-Spezifika auf der Orchestrierungsschicht verfügbar machen. So hat beispielsweise Microsoft sein Cloud-Angebot für Container-Cluster um die Verfügbarkeit von SGX-fähigen Knoten erweitert und stellt u.a. ein Plugin zur Verwaltung des für containerisierte Anwendungen verfügbaren EPC-Speicher bereit [Mic20].

5.4 Zusammenfassung

Wie in den vorangehenden Abschnitten gezeigt wurde, ist die Sicherung von Integrität und Vertraulichkeit sensitiver Daten einer containerisierten Anwendung bzw. Multi Service Application grundsätzlich möglich und weist starke Analogien zur Migration herkömmlicher Programme auf.
Sowohl im wissenschaftlichen Umfeld als auch von Privatunternehmen werden zu diesem Thema Ansätze entwickelt, die den Umgang mit Containern vereinfachen sollen. Dennoch können alle in dieser Arbeit vorgestellten Studien als für die Migration containerisierter Anwendungen geeignet betrachtet werden, da der Einfluss der Containerumgebung auf die Durchführung der eigentlichen Portierung, wie in Abschnitt 5.2 gezeigt, gering ist.
Die Verwendung der generischen Methodik zur Portierung von Multi Service Applications kann auch in diesem Fall verwendet werden, wobei insbesondere im ersten (Abschnitt 3.1) und letzten (Abschnitt 3.4) Schritt die Spezifika der Containervirtualisierung von Bedeutung sind. So müssen bei der Analyse der Anwendungsstruktur die in dem jeweiligen Image enthaltenen Programmdateien und Abhängigkeiten bestimmt werden und die Implementierung der Migration wird gänzlich innerhalb der verwendeten Container durchgeführt.
Zuletzt wurde gezeigt, dass auch der Einsatz von Werkzeugen zur Orchestrierung von Container-Clustern nicht dadurch verhindert wird, dass die containerisierten Anwendungen mittel SGX geschützt werden. Weiterhin sind verschiedene Möglichkeiten zur Unterstützung bzw. Automatisierung des Deployments einer Multi Service Application, welche mittels Containern bereitgestellt und in Enklaven ausgeführt wird, denkbar.

6 Fazit

6.1 Zusammenfassung

Die vorliegende Masterarbeit thematisiert die Verwendung der Intel Software Guard Extensions zur Integritäts- und Vertraulichkeitssicherung besonders schützenswerter Anwendungsdaten während ihrer Verarbeitung durch den Prozessor (*Data in use*). Insbesondere wird der Fokus auf die Ausführung solche Daten verarbeitenden Codes auf einem entfernten Rechner, beispielsweise in Public-Cloud-Umgebungen, gelegt. Weiterhin steht das Erreichen dieses Schutzes bei bereits vorhandenen (serviceorientierten) Anwendungen mit möglichst geringem Aufwand im Vordergrund.

Um ein Verständnis der Befehlssatzerweiterung für Intel-Prozessoren aufzubauen, werden zuerst fundamentale SGX-Eigenschaften und Funktionsweisen sowie die zugrundeliegenden Prinzipien erläutert. Anschließend erfolgt die Kategorisierung und ein Vergleich von 13 unterschiedlichen, veröffentlichten Studien über die Migration von bestehenden Applikationen und des offiziellen Vorgehens zur Neuentwicklung von SGX-Anwendungen. Zusätzlich wird ein Überblick über verwandte, kommerzielle Angebote gegeben.
Um insbesondere die Anpassung von komplexen Multi Service Applications zum Einsatz in sicheren Enklaven zu erleichtern, wurde eine strukturierte Methodik entwickelt, welche von den vorhandenen Migrationsansätzen Gebrauch macht. Sie beinhaltet vier Schritte zur Analyse der Anwendung und Planung, Vorbereitung sowie Implementierung der eigentlichen Migration mit reduzierter Komplexität.
Zur Veranschaulichung des Vorgehens und Evaluation ihrer praktischen Anwendbarkeit wurde eine beispielhafte, nicht für SGX entwickelte, Webanwendung mittels dieser Methodik migriert und die jeweils unternommenen Schritte und getroffenen Entscheidungen erläutert. Anhand von verschiedenen Makro-Benchmarks wird gezeigt, dass die Gesamtperformanz der Anwendung durch die Migration nachteilig beeinflusst wird. Je nach Anwendungsszenario muss hierbei eine eineinhalb- bis zehnfache Verarbeitungsdauer sowie eine Limitierung bezüglich paralleler Zugriffe und maximaler Dateigrößen in Kauf genommen werden.
Abschließend wird die Möglichkeit der Nutzung von SGX durch in Containern ausgeführte Anwendungen diskutiert, sowie die in diesem Szenario zu beachtenden Besonder-

heiten herausgestellt. Dabei wird deutlich, dass der Einfluss einer Containerumgebung auf die Durchführung der Migration gering ist und die vorgestellte Methodik – bis auf wenige Anpassungen – anwendbar bleibt. Auch die Verwendung von Orchestrierungswerkzeugen zur Containerverwaltung ist ohne Einschränkungen möglich, ihre Weiterentwicklung bietet außerdem Potential für zusätzliche Automatisierung.

Das Ziel der vereinfachten Migration von aus mehreren Diensten bestehenden Anwendungen zum Schutz sensitiver *Data in use* durch Intel SGX wurde folglich – mit den in Abschnitt 4.3 beschriebenen Einschränkungen – erreicht.

6.2 Ausblick

Ausgehend von der in dieser Arbeit vorgestellten Methodik, bietet sich die Durchführung weiterführender Studien an.
Obwohl Intel SGX die verbreitetste und (zum Zeitpunkt der Erstellung dieser Arbeit) umfangreichste Implementierung einer TEE darstellt, wurden verschiedene weitere Technologien mit ähnlichen Schutzzielen vorgestellt. Insbesondere kann die Verwendung von „Overshadow“ [Che+08], „SecureME“ [Chh+11], AMD „Memory Encryption“ [KPW16], ARM „TrustZone“ [Nga+16] oder „Sanctum“ [CLD16] sowie die Portierung einer bestehenden Anwendung zur Nutzung der jeweiligen TEE untersucht werden. Ein Vergleich zu Intel SGX bietet sich z.B. bezüglich der Sicherheit, Performanz und des Nutzungsaufwands an.
Im Hinblick auf die starke Vergrößerung des EPC-Speichers in den zukünftigen CPU-Versionen sollte untersucht werden, welche Veränderungen der Einsatz eines solchen Prozessors bezüglich der Ausführungsgeschwindigkeit und Sicherheit mit sich bringt. Insbesondere die Notwendigkeit einer Anwendungspartitionierung muss neu bewertet werden.
Außerdem ist eine weitere Untersuchung und praktische Evaluation der von einigen Migrationsansätzen bereitgestellten Möglichkeiten bezüglich der Portierung von Container-Anwendungen sinnvoll. Unter Nutzung dieser Arbeit als Grundlage bietet sich die Entwicklung einer generischen – und insbesondere von den verwendeten Images möglichst unabhängigen – Methodik für die Migration solcher Anwendungen an.
Darauf aufbauend sollte die Orchestrierung von Containern mit SGX-Anwendungen in der Praxis getestet werden. Weiterhin ist die Untersuchung von Optionen zur möglichst automatisierten Konfiguration – und evtl. Migration – von containerbasierten (serviceorientierten) Applikationen sinnvoll.

Anhang A: Vergleich der Migrationsansätze

In diesem Kapitel werden die in der Forschung existierenden Ansätze zur Migration von Nicht-SGX-Anwendungen nach den ihnen zugrundeliegenden Migrationsstrategien (siehe Abschnitt 2.2.1) gruppiert und tabellarisch gegenübergestellt.
Dabei kommen die in Abschnitt 2.2.2 erläuterten Kriterien als Vergleichsgrundlage zum Einsatz, wobei die Kompatibilität in Bezug auf Systemcalls als „Syscall-Komp.“ abgekürzt wird. Weiterhin nutzen einige Ansätze den Systemspeicher außerhalb des EPC zum Datenaustausch zwischen Enklave und $\mathcal{P}_u$. Dieser wird nachfolgend als „untrusted Memory“ (kurz $\mathcal{M}_u$) bezeichnet.
Informationen, welche mit Hilfe der vorliegenden Studien sowie weiterführender Recherchen nicht bestimmt werden konnten, werden in den folgenden Tabellen mit einem Fragezeichen gekennzeichnet.

Name	**TCB**	**Open Source**	**Aufwand**	**Syscall-Strategie**	**Syscall-Komp.**	**Besonderheiten**	**Einschränkungen**
Intel SGX SDK [Int20k]	?	✓	sehr hoch	Standard	komplett	manuelle Programmierung, Generierung der Bridge Functions durch das *Edger8r*-Werkzeug	keine
Switchless Calls [Tia+18]	?	✓	gering(1)	Workerpool(2) mit bedarfsgerechter Anpassung der Poolgröße	komplett	im Intel SGX SDK seit 2018 als optionaler Standard integriert	keine
Eleos [Ore+17]	≈ 1 kLOC	✓	sehr hoch(3)	Workerpool(2) mit Steuerung durch Remote Procedure Calls	?	Möglichkeit zur Verwaltung von EPC-Speicher in der Enklave durch eine zusätzliche Abstraktionsschicht („Secure User-managed Virtual Memory“)	modifizierter SGX-Treiber notwendig; nur in C++ verfügbar

(1) Aufwand zum Ersetzen der Standardprozedur für Systemcalls durch Switchless Calls

(2) Statt einen Systemcall durch Verlassen der Enklave auszuführen, wird ein Pool von Anwendungsthreads im $\mathcal{P}_u$ verwendet, um die Funktion asynchron aufzurufen und Funktionsparameter und Rückgabewerte über einen gemeinsamen Speicherbereich außerhalb des EPC ($\mathcal{M}_u$) mit der Enklave auszutauschen.

(3) Quellcodemodifikationen zur Verwendung spezifischer Eleos-Funktionen notwendig

Tabelle A.1: Programmierbibliotheken zur Migration von Nicht-SGX-Anwendungen

Name	**Aufgabe**	**TCB**	**Open Source**	**Aufwand**	**Syscall-Strategie**	**Syscall-Komp.**	**Besonderheiten**	**Einschränkungen**
Glamdring [Lin+17]	Partitionierung in $\mathcal{P}_u$ und $\mathcal{P}_t$	≈ 6 kLOC	✗	hoch(1)	Standard	komplett	automatische Detektion aller Verwendungen der annotierten Variablen und Generierung des entsprechenden *tCodes*	nur für C-Programme geeignet
Panoply [Shi+17]	Aufteilung in funktionale Bestandteile zur TCB-Minimierung	≈ 20 kLOC	✓	hoch(2)	teils durch Panoply in Enklave, teils Standard	fast komplett(3)	Absicherung des Datenflusses zwischen mehreren funktionalen Einheiten („Microns“); dynamisches Multithreading in Enklaven	modifizierter SGX-Treiber und SDK für Multithreading erforderlich
HotCalls [WBA17]	schnellere Systemcalls	≈ 150 LOC	✗	gering(4)	einzelner $\mathcal{P}_u$-Workerthread für Syscall-Ausführung, Kommunikation über $\mathcal{M}_u$	komplett	(teil-) automatisierte Erkennung verwendeter Systemcalls und Generierung von Code zu deren schnelleren Ausführung (als sog. „HotCalls“)	keine

Tabelle A.2: Automatisierungsframeworks zur Migration von Nicht-SGX-Anwendungen (Fortsetzung folgt)

Name	Aufgabe	TCB	Open Source	Aufwand	Syscall-Strategie	Syscall-Komp.	Besonderheiten	Einschränkungen
lxcsgx [Tia+19]	Partitionierung in $\mathcal{P}_u$ und $\mathcal{P}_t$	≈ 120 kLOC	✗	hoch(1)	Standard	komplett	Ausführung von A_{SGX} im Container mit EPC-Quota(5); teilautomatisierte Partitionierung durch Compiler-Plugin; Bereitstellung eines Software Trusted Platform Module für andere Enklaven	modifizierter Linux-Kernel benötigt; unterstützt nur das LXC-Containerformat; Partitionierung nur mittels *GCC*-Compiler (C/C++) möglich

(1) Quelltextannotationen von schützenswerten Variablen notwendig

(2) Quelltextannotationen von Funktionen zur Zuteilung zu „Microns“ (als Enklave oder normale Prozesse ausgeführte Anwendungsbestandteile) sowie Neukompilieren zum Generieren des SGX-Codes notwendig.

(3) 254 von über 300 Systemcalls unterstützt

(4) teilweise manuelle Korrektur bei automatisierter Systemcalldetektion erforderlich

(5) Eingrenzung des maximalen EPC-Speicherbedarfs einer Enklave beim Containerstart möglich

Tabelle A.2: Automatisierungsframeworks zur Migration von Nicht-SGX-Anwendungen (Fortsetzung)

Name	**TCB**	**Open Source**	**Aufwand**	**Syscall-Strategie**	**Syscall-Komp.**	**Besonderheiten**	**Einschränkungen**
SCONE [Arn+16]	≈ 97/187 kLOC[(1)]	✗	gering[(2)]	asynchron durch Workerthreads in SCONE-Kernelmodul	komplett (*musl*)	Migration von Anwendungen in Containern; transparente Verschlüsselung durch *Shields*[(3)] möglich; dynamisches Multithreading in Enklaven; Verfügbarkeit bereits portierter Anwendungen („SCONE Curated Images“ [SCO20d])	nur Docker-Containerformat unterstützt; Remote Attestation muss über SCONE-Service statt IAS erfolgen
Script Shield [Wan+19]	≈ 700 + 80.000 LOC[(4)]	✓	hoch bzw. sehr gering[(5)]	Standard mit zusätzlichen Sicherheitschecks	eingeschränkt[(6)] (*musl*)	Migration von interpretierten Anwendungen durch Portierung des Interpreters in eine Enklave und Ausführen der Programmdateien	sichere Bereitstellung der Quelltexte notwendig; bisher nur Lua-, JavaScript- und Squirrel-Interpreter portiert

[(1)] Größe der SCONE-Variante ohne bzw. mit aktiviertem Shielding (jeweils inklusive der benötigten Bibliothek *musl libc* mit ca. 80 kLOC)

[(2)] Neukompilieren der A_L mit SCONE-Compiler sowie ggf. Spezifikation der verwendeten *Shields* notwendig

[(3)] *Shields* für ein- und ausgehenden Netwerkverkehr, Dateien und `stdin`/`stdout`-Streams aktivierbar

[(4)] ScriptShield selbst umfasst nur ca. 700 LOC, zusätzlich wird allerdings noch die Standard-C-Bibliothek *musl libc* benötigt, die ca. 80 kLOC umfasst.

[(5)] Aufwand für die Portierung des Interpreters bzw. der skriptbasierten Anwendung

[(6)] 60 von über 300 Systemcalls unterstützt

Tabelle A.3: Shielding Layers zur Migration von Nicht-SGX-Anwendungen

Name	TCB	Open Source	Aufwand	Syscall-Strategie	Syscall-Komp.	Besonderheiten	Einschränkungen
Haven [BPH15]	> 5.600 kLOC	✗	sehr gering	möglichst intern(1)	komplett	spezifisch für Windows-Anwendungen entwickelt(2)	Prototyp unterstützt keine Remote Attestation
Graphene-SGX [Gra19c; TPV17]	≈ 1.353 kLOC(3)	✓	sehr gering(4)	möglichst intern(1)	komplett (*glibc*)	Unterstützung von multiplen Enklavenprozessen, „Switchless Calls“ & Sealing; dynamisches Nachladen von Bibliotheken mit Integritätschecks; Migration von Containern möglich	modifizierter SGX-Treiber oder Kernelpatches benötigt; nur Docker-Containerformat unterstützt
SGX Kernel [Tia+17]	≈ 87 kLOC(5)	✗	mittel(6)	asynchron durch Workerpool im $\mathcal{P}_u$, Kommunikation über $\mathcal{M}_u$	eingeschränkt(7) (*musl*)	dynamisches Multithreading in Enklaven	keine
SGX-LKL [Pri+20]	≈ 19.000 kLOC(5)	✓	gering(8)	möglichst intern(1), nur 7 Low-Level-Syscalls weitergereicht	komplett (*musl*)	maximale Reduktion der externen Schnittstelle und Informationspreisgabe der Enklave durch Nachbildung von OS-Funktionalitäten im $\mathcal{P}_t$ (9); Portierung von Containerimages statt Anwendungen	Inkaufnahme schlechterer Performanz für höhere Sicherheit, nur Docker-Containerformat unterstützt

Tabelle A.4: Library-OS-Ansätze zur Migration von Nicht-SGX-Anwendungen (Fortsetzung folgt)

Name	TCB	Open Source	Aufwand	Syscall-Strategie	Syscall-Komp.	Besonderheiten	Einschränkungen
Occlum [She+20], [Occ20]	≈ 15 kLOC	✓	gering[10]	möglichst intern[1]	eingeschränkt[11] (*musl*)	sicheres Multitasking in der Enklave mit Prozessisolation; Verifikation der $A_{L,p}$ vor dem Start; transparent verschlüsseltes Dateisystem in der Enklave; Ausführung migrierter Anwendungen in Containern möglich	(bisher) nur statisches Linken unterstützt; benötigt spezielles Kernelmodul; nur Docker-Containerformat unterstützt

[1] wenn möglich, durch das libOS in der Enklave bearbeitet, sonst nach dem Standard-Vorgehen an den $\mathcal{P}_u$ weitergereicht

[2] Haven basiert auf dem „Drawbridge“-Library OS, einer Untermenge des Windows-8-Betriebssystems.

[3] beinhaltet die *glibc*-Bibliothek mit ca. 1.300 kLOC

[4] anwendungsspezifische Graphene-Konfiguration in *Manifest*-Datei notwendig, danach unmodifizierte Ausführung der A_L möglich

[5] beinhaltet die *musl libc*-Bibliothek mit ca. 80 kLOC

[6] theoretisch unmodifizierte Ausführung der A_L, im Beispiel der Studie aber Codemodifikationen erforderlich

[7] 102 von über 300 Systemcalls unterstützt

[8] Erstellung eines verschlüsselten *Disk Images* der A_L durch SGX-LKL-Tool notwendig

[9] Nachbildung zahlreicher Kernel-Funktionalitäten im User Space mit Hilfe der „Linux Kernel Library“ (kurz: LKL) [PGT10]

[10] anwendungsspezifische Occlum-Konfiguration in JSON-Datei und Rekompilieren mit Occlum-Compiler notwendig, danach unmodifizierte Ausführung möglich

[11] z.B. `fork()`-Systemcall kann nicht unterstützt werden

Tabelle A.4: Library-OS-Ansätze zur Migration von Nicht-SGX-Anwendungen (Fortsetzung)

Anhang B: Details der WordPress-Migration

Während der beispielhaften Migration der Multi Service Application „WordPress“ werden diverse Konfigurations-, Skript- und weitere Dateien erstellt. Zur besseren Nachvollziehbarkeit werden deren relevante Inhalte in diesem Kapitel bereitgestellt. Zusätzlich erfolgt eine Auflistung aller Abhängigkeiten der einzelnen A_{MS}-Dienste.

B.1 Abhängigkeiten der WordPress-Dienste

In diesem Abschnitt werden alle während des Migrationsschrittes 1 (Abschnitt 4.1.1) bestimmten Abhängigkeiten der drei WordPress-Dienste aufgeführt.
Der Installationspfad der jeweiligen Anwendung wird dabei im Folgenden mit `<I>` abgekürzt.

B.1.1 Webserver (Lighttpd)

Programmdatei

- `<I>/sbin/lighttpd`

Bibliotheken

- shared libraries (nach `ldd`)
  - `/lib/x86_64-linux-gnu/libdl.so.2`
  - `/lib/x86_64-linux-gnu/libc.so.6`
  - `/usr/lib/x86_64-linux-gnu/libcrypto.so.1.1`
  - `/lib/x86_64-linux-gnu/libpthread.so.0`
  - `/lib64/ld-linux-x86-64.so.2`
- *Fortsetzung folgt*

Bibliotheken *(Fortsetzung)*

- Lighttpd-Module (laut Konfigurationsdatei)
  - `<I>/lib/mod_access.so`
  - `<I>/lib/mod_accesslog.so`
  - `<I>/lib/mod_fastcgi.so`
  - `<I>/lib/mod_openssl.so`
- Weitere Bibliotheken (durch Versuch und Irrtum)[1]
  - `<I>/lib/mod_indexfile.so`
  - `<I>/lib/mod_dirlisting.so`
  - `<I>/lib/mod_staticfile.so`

Dateien

- Konfigurationsdateien
  - `lighttpd.conf`
  - `lighttpd-generic.conf`
  - `lighttpd-server.conf`
  - `lighttpd-fastcgi-php.conf`
  - `lighttpd-ssl.conf`
- Log- und Zertifikatsdateien (laut Konfigurationsdatei)
  - `/var/log/lighttpd/error.log`
  - `/var/log/lighttpd/access.log`
  - `/etc/lighttpd/lighttpd.pem`[2]
- WebApp-Dateien
  - alle (PHP-) Dateien für WordPress
- Weitere Dateien (durch Versuch und Irrtum)
  - `/var/tmp/`[3]
  - `/proc/loadavg`[4]

[1] Die drei Lighttpd-Module `mod_indexfile`, `mod_dirlisting` und `mod_staticfile` sind Standardmodule, die in jedem Fall automatisch geladen werden.

[2] Für HTTPS-Verbindungen benötigt der Webserver je ein Zertifikat für sich selbst und die entsprechende Certificate Authority sowie seinen privaten Schlüssel. Die Datei `lighttpd.pem` beinhaltet alle drei Informationen.

[3] Das Verzeichnis wird als temporärer Speicherort für hochgeladene Dateien benötigt.

[4] Diese Datei stellt eine Schnittstelle zum Kernel dar und beinhaltet Informationen über die aktuelle Systemauslastung [Qui+17].

B.1.2 Anwendungsserver (PHP-FPM)

Programmdatei

- `<I>/bin/php-fpm`

Bibliotheken

- shared libraries (nach `ldd`)
  - `/lib/x86_64-linux-gnu/libresolv.so.2`
  - `/lib/x86_64-linux-gnu/librt.so.1`
  - `/lib/x86_64-linux-gnu/libm.so.6`
  - `/lib/x86_64-linux-gnu/libdl.so.2`
  - `/usr/lib/x86_64-linux-gnu/libxml2.so.2`
  - `/usr/lib/x86_64-linux-gnu/libssl.so.1.1`
  - `/usr/lib/x86_64-linux-gnu/libcrypto.so.1.1`
  - `/lib/x86_64-linux-gnu/libz.so.1`
  - `/usr/lib/x86_64-linux-gnu/libcurl.so.4`
  - `/usr/lib/x86_64-linux-gnu/libpng16.so.16`
  - `/usr/lib/x86_64-linux-gnu/libonig.so.4`
  - `/usr/lib/x86_64-linux-gnu/libsodium.so.23`
  - `/usr/lib/x86_64-linux-gnu/libzip.so.4`
  - `/lib/x86_64-linux-gnu/libc.so.6`
  - `/lib/x86_64-linux-gnu/libpthread.so.0`
  - `/usr/lib/x86_64-linux-gnu/libicuuc.so.60`
  - `/lib/x86_64-linux-gnu/liblzma.so.5`
  - `/usr/lib/x86_64-linux-gnu/libnghttp2.so.14`
  - `/usr/lib/x86_64-linux-gnu/libidn2.so.0`
  - `/usr/lib/x86_64-linux-gnu/librtmp.so.1`
  - `/usr/lib/x86_64-linux-gnu/libpsl.so.5`
  - `/usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2`
  - `/usr/lib/x86_64-linux-gnu/libldap_r-2.4.so.2`
  - `/usr/lib/x86_64-linux-gnu/liblber-2.4.so.2`
  - *Fortsetzung folgt*

- shared libraries (nach `ldd`) *(Fortsetzung)*
  - `/usr/lib/x86_64-linux-gnu/libicudata.so.60`
  - `/usr/lib/x86_64-linux-gnu/libstdc++.so.6`
  - `/lib/x86_64-linux-gnu/libgcc_s.so.1`
  - `/usr/lib/x86_64-linux-gnu/libunistring.so.2`
  - `/usr/lib/x86_64-linux-gnu/libgnutls.so.30`
  - `/usr/lib/x86_64-linux-gnu/libhogweed.so.4`
  - `/usr/lib/x86_64-linux-gnu/libnettle.so.6`
  - `/usr/lib/x86_64-linux-gnu/libgmp.so.10`
  - `/usr/lib/x86_64-linux-gnu/libkrb5.so.3`
  - `/usr/lib/x86_64-linux-gnu/libk5crypto.so.3`
  - `/lib/x86_64-linux-gnu/libcom_err.so.2`
  - `/usr/lib/x86_64-linux-gnu/libkrb5support.so.0`
  - `/usr/lib/x86_64-linux-gnu/libsasl2.so.2`
  - `/usr/lib/x86_64-linux-gnu/libgssapi.so.3`
  - `/usr/lib/x86_64-linux-gnu/libp11-kit.so.0`
  - `/usr/lib/x86_64-linux-gnu/libtasn1.so.6`
  - `/lib/x86_64-linux-gnu/libkeyutils.so.1`
  - `/usr/lib/x86_64-linux-gnu/libheimntlm.so.0`
  - `/usr/lib/x86_64-linux-gnu/libkrb5.so.26`
  - `/usr/lib/x86_64-linux-gnu/libasn1.so.8`
  - `/usr/lib/x86_64-linux-gnu/libhcrypto.so.4`
  - `/usr/lib/x86_64-linux-gnu/libroken.so.18`
  - `/usr/lib/x86_64-linux-gnu/libffi.so.6`
  - `/usr/lib/x86_64-linux-gnu/libwind.so.0`
  - `/usr/lib/x86_64-linux-gnu/libheimbase.so.1`
  - `/usr/lib/x86_64-linux-gnu/libhx509.so.5`
  - `/usr/lib/x86_64-linux-gnu/libsqlite3.so.0`
  - `/lib/x86_64-linux-gnu/libcrypt.so.1`
  - `/lib64/ld-linux-x86-64.so.2`
- PHP-Erweiterungen (laut Konfigurationsdatei)
  - *keine*[5]
- *Fortsetzung folgt*

[5] Da die PHP-Anwendung mit statisch gelinkten Bibliotheken kompiliert wurde, entfällt die Notwendigkeit zum Nachladen von dynamischen Extensions zur Laufzeit.

Bibliotheken *(Fortsetzung)*

- Weitere Bibliotheken (durch Versuch und Irrtum)
  - `/lib/x86_64-linux-gnu/libnss_compat.so.2`
  - `/lib/x86_64-linux-gnu/libnss_systemd.so.2`
  - `/lib/x86_64-linux-gnu/libnss_files.so.2`
  - `/lib/x86_64-linux-gnu/libnss_mdns4_minimal.so.2`
  - `/lib/x86_64-linux-gnu/libnss_dns.so.2`
  - `/lib/x86_64-linux-gnu/libnss_myhostname.so.2`

Dateien

- Konfigurationsdateien
  - `php-fpm.conf`
  - `php-fpm.d/www.conf`
  - `php.ini`
- Logdateien (laut Konfigurationsdatei)[6]
  - `/var/log/php/fpm.log`
  - `/var/log/php/php.log`
- WebApp-Dateien
  - alle (PHP-) Dateien für WordPress
  - Datenverzeichnisse innerhalb des WordPress-Ordners (z.B. für Uploads)
- Weitere Dateien (durch Versuch und Irrtum)
  - `/etc/nsswitch.conf`
  - `/etc/passwd`
  - `/etc/group`
  - `/etc/host.conf`
  - `/etc/resolv.conf`
  - `/etc/hosts`
  - `/etc/gai.conf`
  - `/tmp`[7]

[6] PHP-FPM verwendet zwei Logdateien; eine für den Prozessmanager und eine für die von ihm gestarteten PHP-FastCGI-Prozesse.

[7] Das temporäre Verzeichnis wird zum Zwischenspeichern von Uploads benötigt, bevor sie in den WordPress-Ordner verschoben werden.

PHP-Erweiterungen Neben diesen Bibliotheken und Dateien benötigt der Anwendungsserver zur Ausführung der Webanwendung WordPress diverse Zusatzfunktionalität, die mittels der in Tabelle B.1 aufgelisteten PHP-Erweiterungen aktiviert wird. Auch die bei der Build-Konfiguration erforderlichen Kommandozeilenparameter werden aufgezeigt.

Erweiterung	**Standard**[1]	**`./configure`-Parameter zur Aktivierung**
`curl`	✗	`--with-curl`
`dom`	✓	-
`exif`	✗	`--enable-exif`
`fileinfo`	✓	-
`filter`	✓	-
`ftp`	✗	`--enable-ftp`
`gd`	✗	`--enable-gd`
`hash`	✓	-
`iconv`	✓	-
`json`	✓	-
`sodium`	✗	`--with-sodium`
`mbstring`	✗	`--enable-mbstring`
`mysqli`	✗	`--with-mysqli`
`openssl`	✗	`--with-openssl`
`pcre`	✓	-
`SimpleXML`	✓	-
`sockets`	✗	`--enable-sockets`
`xml`	✓	-
`xmlreader`	✓	-
`zip`	✗	`--with-zip`
`zlib`	✗	`--with-zlib`

[1] Diese Erweiterungen werden automatisch aktiviert, wenn dies durch entsprechende `./configure`-Parameter nicht explizit unterbunden wird.

Tabelle B.1: Für WordPress benötigte PHP-Erweiterungen

B.1.3 Datenbank (MariaDB)

Programmdatei

- `<I>/bin/mariadbd`

Bibliotheken

- shared libraries (nach `ldd`)
  - `/usr/lib/x86_64-linux-gnu/libpcre2-8.so.0`
  - `/lib/x86_64-linux-gnu/libcrypt.so.1`
  - `/lib/x86_64-linux-gnu/libz.so.1`
  - `/usr/lib/x86_64-linux-gnu/libssl.so.1.1`
  - `/usr/lib/x86_64-linux-gnu/libcrypto.so.1.1`
  - `/lib/x86_64-linux-gnu/libpthread.so.0`
  - `/lib/x86_64-linux-gnu/libdl.so.2`
  - `/usr/lib/x86_64-linux-gnu/libstdc++.so.6`
  - `/lib/x86_64-linux-gnu/libm.so.6`
  - `/lib/x86_64-linux-gnu/libgcc_s.so.1`
  - `/lib/x86_64-linux-gnu/libc.so.6`
  - `/lib64/ld-linux-x86-64.so.2`
- MariaDB-Plugins (laut Konfigurationsdatei)
  - *keine*
- Weitere Bibliotheken (durch Versuch und Irrtum)
  - `/lib/x86_64-linux-gnu/libnss_compat.so.2`
  - `/lib/x86_64-linux-gnu/libnss_systemd.so.2`
  - `/lib/x86_64-linux-gnu/libnss_files.so.2`
  - `/lib/x86_64-linux-gnu/librt.so.1`

Dateien

- Konfigurationsdateien
  - `my.cnf`
  - `my.cnf.d/mariadb-server.cnf`
- *Fortsetzung folgt*

Dateien *(Fortsetzung)*

- Log- und Laufzeitdateien (laut Konfigurationsdatei)
  - `/run/mysqld/mysqld.sock`
  - `/run/mysqld/mysqld.pid`
  - `/var/lib/mysql/mysql.err`
- Verzeichnis für temporäre Daten (laut Konfigurationsdatei)
  - `/var/tmp/`
- Dateien des Backing Storage
  - `/var/lib/mysql/`[8]
- Weitere Dateien (durch Versuch und Irrtum)
  - `/etc/nsswitch.conf`
  - `/etc/passwd`
  - `/etc/group`
  - `/etc/localtime`

B.2 Dateien und Befehle

B.2.1 MariaDB

```
1  # Bei Bedarf: Installation von OpenSSL
2  sudo apt install openssl
3
4  mkdir db-certs ca
5  # Zertifikat und Schlüssel für CA generieren
6  openssl genrsa 2048 > ca/ca-key.pem
7  openssl req -sha1 -new -x509 -nodes -days 3650 -key ca/ca-key↩
     ↪.pem > ca/ca-cert.pem
8  # Zertifikat und Schlüssel für MariaDB generieren
9  # Hinweis: anderen "Common Name" als für CA wählen!
10 openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout ↩
     ↪db-certs/server-key.pem > db-certs/server-req.pem
11 # Signatur des Server- und Kopie des CA-Zertifikates
```

[8] In diesem Verzeichnis speichert MariaDB alle Inhalte und Metadaten über die Datenbank.

```
openssl x509 -sha1 -req -in db-certs/server-req.pem -days 730 -CA ca/ca-cert.pem -CAkey ca/ca-key.pem -set_serial 01 > db-certs/server-cert.pem
cp ca/ca-cert.pem db-certs/ca-cert.pem
rm db-certs/server-req.pem
# Zugriffsrechte für MySQL-Nutzer setzen
sudo chown 100:101 db-certs/*
```

Listing B.1: Generieren von Zertifikaten für MariaDB

```
#
# Konfigurationsdatei für den MariaDB Server
#

# Abschnitt für den Server-Dienst "mariadbd"
[server]

# Grundlegende Einstellungen
port              = 3306  # Standard-Port für MySQL / MariaDB
bind-address      = 0.0.0.0  # akzeptiere Anfragen von allen IP-Adressen
socket            = /run/mysqld/mysqld.sock
pid-file          = /run/mysqld/mysqld.pid
datadir           = /var/lib/mysql
tmpdir            = /var/tmp

# Performanz-Einstellung
thread_handling   = pool-of-threads

# Logdatei für Fehler
log_error         = /var/lib/mysql/mysql.err
expire_logs_days  = 10
max_binlog_size   = 100M

# TLS-Funktionalität aktivieren
ssl-ca            = /etc/mysql-ssl/ca-cert.pem
ssl-cert          = /etc/mysql-ssl/server-cert.pem
ssl-key           = /etc/mysql-ssl/server-key.pem
tls_version       = TLSv1.2,TLSv1.3
```

Listing B.2: Inhalt der MariaDB-Datei `mariadb-server.cnf`

```
# nach Abschluss der Datenbasis-Initialisierung:
sudo mv database input
mkdir database
# Starten des SCONE-Containers
docker run --rm -it \
  -v $PWD/database:/var/lib/mysql \
  -v $PWD/input:/input \
  -v $PWD:/output \
    "sconecuratedimages/crosscompilers"

# im Container:
# leeres FSPF erstellen
scone fspf create db.pb
# Spezifiziere Wurzelverzeichnis als nicht verschlüsselt
scone fspf addr db.pb / --not-protected --kernel /
# Spezifiziere Datenbasis als verschlüsselt
scone fspf addr db.pb /var/lib/mysql --encrypted --kernel /↩
   ↪var/lib/mysql
# Verschlüsselung des DB-Verzeichnisses
scone fspf addf db.pb /var/lib/mysql /input /var/lib/mysql
# Verschlüsselung des FSPF
scone fspf encrypt db.pb > keytag
mv db.pb keytag /output/
exit

# Zugriffsrechte für MySQL-Nutzer setzen
sudo chown -R 100:101 database
sudo rm -r input
```

Listing B.3: Erstellung des SCONE File System Protection Files für MariaDB

B.2.2 Lighttpd

```
#
# Makefile für Lighttpd in Graphene-SGX
#

# Variablendefinitionen
THIS_DIR := $(dir $(lastword $(MAKEFILE_LIST)))
INSTALL_DIR ?= $(THIS_DIR)install
```

```
8
9  LIGHTTPD_SRC ?= $(THIS_DIR)lighttpd-1.4.54
10 LIGHTTPD_HASH ?= "5151d38cb7c4c40effa13710e77ebd\
11 bef899f945b062cf32befc02d128ac424c"
12 LIGHTTPD_MIRRORS ?= \
13   https://download.lighttpd.net/lighttpd/releases-1.4.x/
14 LIGHTTPD_EXECUTABLE = $(INSTALL_DIR)/sbin/lighttpd
15
16 WEBROOT_DIR ?= <WPDir>
17
18 # Relativer Pfad zum <GDir>
19 GRAPHENEDIR ?= $(THIS_DIR)../..
20 # Schlüssel zur Enklavensignatur
21 SGX_SIGNER_KEY ?= $(GRAPHENEDIR)/↩
      ↪Pal/src/host/Linux-SGX/signer/enclave-key.pem
22
23 # Debugging-Ausgaben von Graphene mit Variable DEBUG=1 ↩
      ↪aktivierbar
24 ifeq ($(DEBUG),1)
25 GRAPHENEDEBUG = inline
26 else
27 GRAPHENEDEBUG = none
28 endif
29
30 GREP = grep
31 SED  = sed
32 LDD  = ldd
33 AWK  = awk
34
35 # Standard-Einstiegspunkt: Bereite alles für Ausführung vor
36 .PHONY: all
37 all: $(LIGHTTPD_EXECUTABLE) lighttpd.manifest pal_loader
38 ifeq ($(SGX),1)
39 all: lighttpd.token
40 endif
41
42 # Standard-Graphene-Variablen definieren
43 include ../../Scripts/Makefile.configs
44
45 # Kompilieren des Webservers
46 $(LIGHTTPD_EXECUTABLE): $(LIGHTTPD_SRC)/configure
```

```
47    cd $(LIGHTTPD_SRC) && ./configure --prefix=$(abspath ↩
      ↪$(INSTALL_DIR)) \
48      --without-pcre --without-zlib --without-bzip2 --↩
      ↪with-openssl
49    cd $(LIGHTTPD_SRC) && $(MAKE)
50    cd $(LIGHTTPD_SRC) && $(MAKE) install
51
52  # WS-Quellcode entpacken
53  $(LIGHTTPD_SRC)/configure: $(LIGHTTPD_SRC).tar.gz
54    tar -xzf $<
55    # Zeitstempel erneuern
56    touch $(LIGHTTPD_SRC)/configure
57
58  # Download des WS-Quellcodes mit Graphene-Hilfsprogramm
59  $(LIGHTTPD_SRC).tar.gz:
60    $(GRAPHENEDIR)/Scripts/download --output $@ --sha256 ↩
      ↪$(LIGHTTPD_HASH) \
61      $(foreach mirror,$(LIGHTTPD_MIRRORS),--url $(mirror)/↩
      ↪$(LIGHTTPD_SRC).tar.gz)
62
63  # Manifest-Datei generieren (Variablen in der Vorlage ↩
      ↪ersetzen)
64  lighttpd.manifest: lighttpd.manifest.template
65    @$(SED) -e 's|$$(GRAPHENEDIR)|'"$(GRAPHENEDIR)"'|g' \
66      -e 's|$$(GRAPHENEDEBUG)|'"$(GRAPHENEDEBUG)"'|g' \
67      -e 's|$$(INSTALL_DIR)|'"$(abspath␣$(INSTALL_DIR))"'|g' \
68      -e 's|$$(ARCH_LIBDIR)|'"$(ARCH_LIBDIR)"'|g' \
69      -e 's|$$(WEBROOT_DIR)|'"$(WEBROOT_DIR)"'|g' \
70      $< > $@
71
72  # SGX-spezifisches Manifest und Enklavensignatur generieren
73  lighttpd.manifest.sgx: lighttpd.manifest ↩
      ↪$(LIGHTTPD_EXECUTABLE)
74    $(GRAPHENEDIR)/Pal/src/host/Linux-SGX/signer/pal-sgx-sign \
75      -libpal $(GRAPHENEDIR)/Runtime/libpal-Linux-SGX.so \
76      -key $(SGX_SIGNER_KEY) \
77      -manifest $< -output $@
78
79  lighttpd.sig: lighttpd.manifest.sgx
80
81  # Token für Enklaveninitialisierung generieren
```

```
82 lighttpd.token: lighttpd.sig
83   $(GRAPHENEDIR)/↩
     ↪Pal/src/host/Linux-SGX/signer/pal-sgx-get-token -output ↩
     ↪$@ -sig $^
84
85 # libOS-Loader verlinken
86 pal_loader:
87   ln -s $(GRAPHENEDIR)/Runtime/pal_loader $@
88
89 # ‘make‘-Target zum vereinfachten Starten des WS
90 .PHONY: start-native-server
91 start-native-server: all
92   $(LIGHTTPD_EXECUTABLE) -D -m $(INSTALL_DIR)/lib -f ↩
     ↪lighttpd-native-server.conf
93
94 .PHONY: start-graphene-server
95 start-graphene-server: all
96   ./pal_loader lighttpd.manifest -D -m $(INSTALL_DIR)/lib -f ↩
     ↪lighttpd.conf
97
98 # Graphene-Daten aufräumen
99 .PHONY: clean
100 clean:
101   $(RM) *.manifest *.manifest.sgx *.token *.sig pal_loader
102
103 # Lighttpd-Dateien löschen
104 .PHONY: distclean
105 distclean: clean
106   $(RM) -r $(LIGHTTPD_SRC).tar.gz $(LIGHTTPD_SRC) \
107     $(INSTALL_DIR) $(TEST_DATA) *.pem
```

Listing B.4: Inhalt der Lighttpd-Datei `Makefile`

```
1 #
2 # Manifest-Vorlage für lighttpd
3 #
4
5 # Die Programmdatei, die in Graphene geladen werden soll
6 loader.exec = file:$(INSTALL_DIR)/sbin/lighttpd
7 loader.argv0_override = lighttpd
8
```

```
# Bibliothek des libOS
loader.preload = file:$(GRAPHENEDIR)/Runtime/libsysdb.so
loader.debug_type = $(GRAPHENEDEBUG)

# Kommandozeilenargumente durchreichen (nicht für ↩
   ↪Produktiveinsatz)
loader.insecure__use_cmdline_argv = 1

# Umgebungsvariablen für den Webserver definieren
# Graphene-Bibliotheken vor weiteren Bibliotheken laden
loader.env.LD_LIBRARY_PATH = /lib:/usr$(ARCH_LIBDIR):$(↩
   ↪INSTALL_DIR)/lib

# im libOS eingebundene Verzeichnisse:
# viele Standard-Bibliotheken stellt Graphene direkt bereit
fs.mount.lib.type = chroot
fs.mount.lib.path = /lib
fs.mount.lib.uri = file:$(GRAPHENEDIR)/Runtime
# weitere Bibliotheken des Host-OS verfügbar machen
fs.mount.lib2.type = chroot
fs.mount.lib2.path = /usr$(ARCH_LIBDIR)
fs.mount.lib2.uri = file:/usr$(ARCH_LIBDIR)
# Installationsverzeichnis von Lighttpd (für Lighttpd-Module)
fs.mount.cwd.type = chroot
fs.mount.cwd.path = $(INSTALL_DIR)
fs.mount.cwd.uri = file:$(INSTALL_DIR)
# für Logdateien
fs.mount.log.type = chroot
fs.mount.log.path = /var/log/lighttpd
fs.mount.log.uri = file:$(INSTALL_DIR)/var/log/lighttpd
# WordPress-Verzeichnis
fs.mount.webroot.type = chroot
fs.mount.webroot.path = $(WEBROOT_DIR)
fs.mount.webroot.uri = file:$(WEBROOT_DIR)
# TLS-Zertifikat
fs.mount.sslcert.type = chroot
fs.mount.sslcert.path = /etc/lighttpd/lighttpd.pem
fs.mount.sslcert.uri = file:lighttpd.pem
# temporärer Ordner für Uploads
fs.mount.var_tmp.type = chroot
fs.mount.var_tmp.path = /var/tmp
```

```
fs.mount.var_tmp.uri = file:/var/tmp
# für /proc/loadavg
fs.mount.proc.type = chroot
fs.mount.proc.path = /proc
fs.mount.proc.uri = file:/proc

# Konfiguration der SGX-Enklave
# max. EPC-Speicher für SGXv1
sgx.enclave_size = 256M
# max. Anzahl an Enklaven-Threads für SGXv1 (2 für Graphene, 1 für Lighttpd)
sgx.thread_num = 3

# in der Enklave benötigte Dateien mit Integritätscheck:
# Glibc-Bibliotheken
sgx.trusted_files.libdl = file:$(GRAPHENEDIR)/Runtime/libdl.so.2
sgx.trusted_files.libc = file:$(GRAPHENEDIR)/Runtime/libc.so.6
sgx.trusted_files.lpthread = file:$(GRAPHENEDIR)/Runtime/libpthread.so.0
sgx.trusted_files.ld = file:$(GRAPHENEDIR)/Runtime/ld-linux-x86-64.so.2
# weitere Bibliotheken
sgx.trusted_files.lssl = file:/usr$(ARCH_LIBDIR)/libssl.so.1.1
sgx.trusted_files.lcrypto = file:/usr$(ARCH_LIBDIR)/libcrypto.so.1.1
# Lighttpd-Module
sgx.trusted_files.mod_access = file:$(INSTALL_DIR)/lib/mod_access.so
sgx.trusted_files.mod_accesslog = file:$(INSTALL_DIR)/lib/mod_accesslog.so
sgx.trusted_files.mod_fastcgi = file:$(INSTALL_DIR)/lib/mod_fastcgi.so
sgx.trusted_files.mod_openssl = file:$(INSTALL_DIR)/lib/mod_openssl.so
sgx.trusted_files.mod_indexfile = file:$(INSTALL_DIR)/lib/mod_indexfile.so
sgx.trusted_files.mod_dirlisting = file:$(INSTALL_DIR)/lib/mod_dirlisting.so
```

```
76 sgx.trusted_files.mod_staticfile = file:$(INSTALL_DIR)/lib/↩
     ↪mod_staticfile.so
77 # Konfigurationsdateien
78 sgx.trusted_files.conf = file:lighttpd.conf
79 sgx.trusted_files.conf2 = file:lighttpd-generic.conf
80 sgx.trusted_files.conf3 = file:lighttpd-server.conf
81 sgx.trusted_files.conf4 = file:lighttpd-fastcgi-php.conf
82 sgx.trusted_files.conf5 = file:lighttpd-ssl.conf
83 # TLS-Zertifikat
84 sgx.trusted_files.sslcert = file:lighttpd.pem
85
86 # in der Enklave benötigte Dateien ohne Integritätscheck:
87 sgx.allowed_files.errorlog = file:$(INSTALL_DIR)/var/log/↩
     ↪lighttpd/error.log
88 sgx.allowed_files.accesslog = file:$(INSTALL_DIR)/var/log/↩
     ↪lighttpd/access.log
89 sgx.allowed_files.webroot = file:$(WEBROOT_DIR)
90 sgx.allowed_files.uploaddir = file:/var/tmp
91 sgx.allowed_files.loadavg = file:/proc/loadavg
```

Listing B.5: Inhalt der *Manifest*-Vorlage für Lighttpd (Datei `lighttpd.manifest.template`)

```
1 #
2 # Konfigurationsdatei lighttpd.conf
3 #
4
5 include "lighttpd-server.conf"
6 include "lighttpd-generic.conf"
7 include "lighttpd-fastcgi-php.conf"
8 include "lighttpd-ssl.conf"
```

Listing B.6: Inhalt der Lighttpd-Datei `lighttpd.conf`

```
1 #
2 # Konfigurationsdatei lighttpd-server.conf
3 #
4
5 # grundlegende Module aktivieren
6 server.modules += ( "mod_access", "mod_accesslog" )
7
```

```
# Webserver konfigurieren
server.document-root        = "<WPDir>"
server.port                 = 8080  # Starten als normaler ↩
   ↪Nutzer moeglich
server.bind                 = "0.0.0.0"

# Logdateien
server.errorlog             = "/var/log/lighttpd/error.log"
accesslog.filename          = "/var/log/lighttpd/access.log"
```

Listing B.7: Inhalt der Lighttpd-Datei `lighttpd-server.conf`

```
#
# Konfigurationsdatei lighttpd-generic.conf
#

# grundlegende Einstellungen
server.event-handler       = "select"
index-file.names           = ( "index.html", "index.php" )

# unterstuetzte Mimetypes festlegen
mimetype.assign  = (
  ".pdf"          =>      "application/pdf",
  ".sig"          =>      "application/pgp-signature",
  ".spl"          =>      "application/futuresplash",
  ".class"        =>      "application/octet-stream",
  ".ps"           =>      "application/postscript",
  ".torrent"      =>      "application/x-bittorrent",
  ".dvi"          =>      "application/x-dvi",
  ".gz"           =>      "application/x-gzip",
  ".pac"          =>      "application/x-ns-proxy-autoconfig",
  ".swf"          =>      "application/x-shockwave-flash",
  ".tar.gz"       =>      "application/x-tgz",
  ".tgz"          =>      "application/x-tgz",
  ".tar"          =>      "application/x-tar",
  ".zip"          =>      "application/zip",
  ".mp3"          =>      "audio/mpeg",
  ".m3u"          =>      "audio/x-mpegurl",
  ".wma"          =>      "audio/x-ms-wma",
  ".wax"          =>      "audio/x-ms-wax",
  ".ogg"          =>      "application/ogg",
```

```
30    ".wav"          =>      "audio/x-wav",
31    ".gif"          =>      "image/gif",
32    ".jpg"          =>      "image/jpeg",
33    ".jpeg"         =>      "image/jpeg",
34    ".png"          =>      "image/png",
35    ".xbm"          =>      "image/x-xbitmap",
36    ".xpm"          =>      "image/x-xpixmap",
37    ".xwd"          =>      "image/x-xwindowdump",
38    ".css"          =>      "text/css",
39    ".html"         =>      "text/html",
40    ".htm"          =>      "text/html",
41    ".php"          =>      "text/html",
42    ".js"           =>      "text/javascript",
43    ".asc"          =>      "text/plain",
44    ".c"            =>      "text/plain",
45    ".cpp"          =>      "text/plain",
46    ".log"          =>      "text/plain",
47    ".conf"         =>      "text/plain",
48    ".text"         =>      "text/plain",
49    ".txt"          =>      "text/plain",
50    ".spec"         =>      "text/plain",
51    ".dtd"          =>      "text/xml",
52    ".xml"          =>      "text/xml",
53    ".mpeg"         =>      "video/mpeg",
54    ".mpg"          =>      "video/mpeg",
55    ".mov"          =>      "video/quicktime",
56    ".qt"           =>      "video/quicktime",
57    ".avi"          =>      "video/x-msvideo",
58    ".asf"          =>      "video/x-ms-asf",
59    ".asx"          =>      "video/x-ms-asf",
60    ".wmv"          =>      "video/x-ms-wmv",
61    ".bz2"          =>      "application/x-bzip",
62    ".tbz"          =>      "application/x-bzip-compressed-tar",
63    ".tar.bz2"      =>      "application/x-bzip-compressed-tar",
64
65  # Standard-Mimetype soll application/octet-stream sein
66    ""              =>      "application/octet-stream",
67  )
```

Listing B.8: Inhalt der Lighttpd-Datei `lighttpd-generic.conf`

```
#
# Konfigurationsdatei lighttpd-fastcgi-php.conf
#

# PHP-Dateien müssen dynamisch behandelt werden
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

# FastCGI Modul aktivieren
server.modules += ( "mod_fastcgi" )

# FastCGI Server konfigurieren
fastcgi.server += ( ".php" =>
        ((
                "host" => "127.0.0.1",
                "port" => 9000,
                "broken-scriptfilename" => "enable"
        ))
)
```

Listing B.9: Inhalt der Lighttpd-Datei `lighttpd-fastcgi-php.conf`

```
#
# Konfigurationsdatei lighttpd-ssl.conf
#

# SSL/TLS-Modul aktivieren
server.modules += ( "mod_openssl" )

# HTTPS aktivieren (auf Port 8443)
$SERVER["socket"] == ":8443" {
        ssl.engine  = "enable"
        ssl.pemfile = "/etc/lighttpd/lighttpd.pem"
}
```

Listing B.10: Inhalt der Lighttpd-Datei `lighttpd-ssl.conf`

```
#
# Konfigurationsdatei lighttpd-native-server.conf
#

# grundlegende Module aktivieren
```

```
server.modules += ( "mod_access", "mod_accesslog" )

# Webserver konfigurieren
server.document-root = "<WPDir>"
server.port          = 8080
server.bind          = "0.0.0.0"

# Logdateien
server.errorlog      = "install/var/log/lighttpd/error.log"
accesslog.filename   = "install/var/log/lighttpd/access.log"

# alle weiteren Konfigurationsdateien importieren
include "lighttpd-generic.conf"
include "lighttpd-fastcgi-php.conf"
include "lighttpd-ssl.conf"
```

Listing B.11: Inhalt der Lighttpd-Datei `lighttpd-native-server.conf`

B.2.3 PHP-FPM

```
#
# Makefile für PHP-FPM in Graphene-SGX
#

# Variablendefinitionen
PHP_SRC = https://github.com/php/php-src.git
PHP_RELEASE = php-7.4.11

SRCDIR = php-src
PHP_EXECUTABLE = $(SRCDIR)/sapi/fpm/php-fpm
CONFIG_PHP = php.ini
CONFIG_FPM = php-fpm.conf
PHP_LOGDIR = log
PHP_USER = elias
PHP_GROUP = elias

# Aktivieren der benötigten Erweiterungen als statische ↩
   ↪Bibliotheken
PHP_CONFIG_OPTIONS = --disable-phpdbg --disable-pdo \
--without-pdo-sqlite --without-sqlite3 \
--with-curl --enable-exif --enable-ftp --enable-gd \
```

```
21 --with-sodium --enable-mbstring --with-mysqli \
22 --with-openssl --enable-sockets --with-zip \
23 --with-zlib --enable-fpm
24
25 WEBROOT_DIR = <WPDir>
26
27 # Relativer Pfad zum <GDir>
28 GRAPHENEDIR = ../..
29 # Schlüssel zur Enklavensignatur
30 SGX_SIGNER_KEY ?= $(GRAPHENEDIR)/↩
      ↪Pal/src/host/Linux-SGX/signer/enclave-key.pem
31
32 # Debugging-Ausgaben von Graphene mit Variable DEBUG=1 ↩
      ↪aktivierbar
33 ifeq ($(DEBUG),1)
34 GRAPHENEDEBUG = inline
35 else
36 GRAPHENEDEBUG = none
37 endif
38
39 MKDIR = mkdir
40
41 # Standard-Einstiegspunkt: Bereite alles für Ausführung vor
42 .PHONY: all
43 all: $(PHP_EXECUTABLE) php-fpm.manifest $(PHP_LOGDIR) ↩
      ↪pal_loader
44 #all: php-fpm.manifest $(PHP_LOGDIR) pal_loader
45 ifeq ($(SGX),1)
46 all: php-fpm.manifest.sgx
47 endif
48
49 # Standard-Graphene-Variablen definieren
50 include ../../Scripts/Makefile.configs
51
52 # Kompilieren von PHP
53 $(PHP_EXECUTABLE): $(SRCDIR)/Makefile
54   $(MAKE) -C $(SRCDIR) -j4
55
56 # Konfigurieren der PHP-Installation
57 $(SRCDIR)/Makefile: $(SRCDIR)/buildconf
58   cd $(SRCDIR) && ./buildconf -f
```

```
59    cd $(SRCDIR) && ./configure $(PHP_CONFIG_OPTIONS)
60
61  # PHP-Quellcode herunterladen
62  $(SRCDIR)/buildconf:
63    git clone $(PHP_SRC) --branch $(PHP_RELEASE) $(SRCDIR)
64
65  # Manifest-Datei generieren (Variablen in der Vorlage ↩
      ↪ersetzen)
66  php-fpm.manifest: php-fpm.manifest.template
67    sed -e 's|$$(GRAPHENEDIR)|'"$(GRAPHENEDIR)"'|g' \
68      -e 's|$$(GRAPHENEDEBUG)|'"$(GRAPHENEDEBUG)"'|g' \
69      -e 's|$$(ARCH_LIBDIR)|'"$(ARCH_LIBDIR)"'|g' \
70      -e 's|$$(WEBROOT_DIR)|'"$(WEBROOT_DIR)"'|g' \
71      -e 's|$$(PHP_EXECUTABLE)|'"$(PHP_EXECUTABLE)"'|g' \
72      $< > $@
73
74  # SGX-spezifisches Manifest und Enklavensignatur generieren
75  php-fpm.manifest.sgx: php-fpm.manifest $(PHP_EXECUTABLE)
76    $(GRAPHENEDIR)/Pal/src/host/Linux-SGX/signer/pal-sgx-sign \
77      -libpal $(GRAPHENEDIR)/Runtime/libpal-Linux-SGX.so \
78      -key $(SGX_SIGNER_KEY) \
79      -manifest $< -output $@ \
80      -exec $(PHP_EXECUTABLE)
81    $(GRAPHENEDIR)/↩
      ↪Pal/src/host/Linux-SGX/signer/pal-sgx-get-token \
82      -output php-fpm.token -sig php-fpm.sig
83
84  # Erstellen des Verzeichnisses für Logdateien
85  $(PHP_LOGDIR):
86    $(MKDIR) $@
87
88  # libOS-Loader verlinken
89  pal_loader:
90    ln -s $(GRAPHENEDIR)/Runtime/pal_loader $@
91
92  # ‘make‘-Targets zum vereinfachten Starten des AS
93  .PHONY: start-native-server
94  start-native-server: all
95    @if test ! -d /var/log/php; then \
96      sudo mkdir /var/log/php ; \
97      sudo chown $(PHP_USER):$(PHP_GROUP) /var/log/php; \
```

```
98      fi
99    @touch /var/log/php/fpm.log
100   @if test -e $(PHP_LOGDIR)/fpm.log; then \
101     rm $(PHP_LOGDIR)/fpm.log; \
102     fi
103   @ln -s /var/log/php/fpm.log $(PHP_LOGDIR)/fpm.log
104   $(PHP_EXECUTABLE) -c $(CONFIG_PHP) -y $(CONFIG_FPM) -F -p .↩
      ↪ -d "error_log=./$(PHP_LOGDIR)/php.log"
105
106 .PHONY: start-graphene-server
107 start-graphene-server: all
108   ./pal_loader php-fpm.manifest -c $(CONFIG_PHP) -y ↩
      ↪$(CONFIG_FPM) -F -p .
109
110 # Graphene-Daten aufräumen
111 .PHONY: clean
112 clean:
113   $(RM) *.token *.sig *.manifest.sgx *.manifest pal_loader
114
115 # Build-Dateien von PHP aufräumen
116 .PHONY: php-clean
117 php-clean:
118   $(MAKE) -C $(SRCDIR) clean
119
120 # alle PHP-Dateien löschen
121 .PHONY: distclean
122 distclean: clean
123   $(RM) -r $(SRCDIR)
```

Listing B.12: Inhalt der PHP-FPM-Datei `Makefile`

```
1  #
2  # Manifest-Vorlage für PHP-FPM
3  #
4
5  # Programmdatei, die in Graphene geladen werden soll
6  # (wird durch Makefile ersetzt)
7  loader.exec = file:$(PHP_EXECUTABLE)
8
9  # Bibliothek des libOS
10 loader.preload = file:$(GRAPHENEDIR)/Runtime/libsysdb.so
```

```
11 loader.debug_type = $(GRAPHENEDEBUG)
12
13 # Kommandozeilenargumente durchreichen (nicht für ↩
   ↪Produktiveinsatz)
14 loader.insecure__use_cmdline_argv = 1
15
16 #loader.insecure__use_host_env = 1
17
18 # Umgebungsvariablen für PHP definieren
19 # Graphene-Bibliotheken vor weiteren Bibliotheken laden
20 loader.env.LD_LIBRARY_PATH = /lib:$(ARCH_LIBDIR):/usr$(↩
   ↪ARCH_LIBDIR)
21
22 # im libOS eingebundene Verzeichnisse:
23 # von Graphene bereitgestellte Standard-Bibliotheken
24 fs.mount.lib.type = chroot
25 fs.mount.lib.path = /lib
26 fs.mount.lib.uri = file:$(GRAPHENEDIR)/Runtime
27 # weitere Bibliotheken des Host-OS
28 fs.mount.lib2.type = chroot
29 fs.mount.lib2.path = $(ARCH_LIBDIR)
30 fs.mount.lib2.uri = file:$(ARCH_LIBDIR)
31 fs.mount.lib3.type = chroot
32 fs.mount.lib3.path = /usr$(ARCH_LIBDIR)
33 fs.mount.lib3.uri = file:/usr$(ARCH_LIBDIR)
34 # Konfigurationsdateien unter /etc
35 fs.mount.etc.type = chroot
36 fs.mount.etc.path = /etc
37 fs.mount.etc.uri = file:/etc
38 # Verzeichnis für Logdateien
39 fs.mount.cwd.type = chroot
40 fs.mount.cwd.path = /var/log/php
41 fs.mount.cwd.uri = file:log
42 # WordPress-Verzeichnis
43 fs.mount.webroot.type = chroot
44 fs.mount.webroot.path = $(WEBROOT_DIR)
45 fs.mount.webroot.uri = file:$(WEBROOT_DIR)
46 # temporärer Ordner für Uploads
47 fs.mount.tmp.type = chroot
48 fs.mount.tmp.path = /tmp
49 fs.mount.tmp.uri = file:/tmp
```

```
50
51 # Konfiguration der SGX-Enklave
52 # max. EPC-Speicher für SGXv1
53 sgx.enclave_size = 512M
54 # max. Anzahl an Enklaven-Threads für SGXv1 (2 für Graphene, ↩
   ↪4 für PHP)
55 sgx.thread_num = 6
56
57 # in der Enklave benötigte Dateien mit Integritätscheck:
58 # Glibc-Bibliotheken
59 sgx.trusted_files.lresolv = file:$(GRAPHENEDIR)/Runtime/↩
   ↪libresolv.so.2
60 sgx.trusted_files.lrt = file:$(GRAPHENEDIR)/Runtime/librt.so↩
   ↪.1
61 sgx.trusted_files.lutil = file:$(GRAPHENEDIR)/Runtime/libutil↩
   ↪.so.1
62 sgx.trusted_files.lm = file:$(GRAPHENEDIR)/Runtime/libm.so.6
63 sgx.trusted_files.ldl = file:$(GRAPHENEDIR)/Runtime/libdl.so↩
   ↪.2
64 sgx.trusted_files.lc = file:$(GRAPHENEDIR)/Runtime/libc.so.6
65 sgx.trusted_files.lpthread = file:$(GRAPHENEDIR)/Runtime/↩
   ↪libpthread.so.0
66 sgx.trusted_files.ld = file:$(GRAPHENEDIR)/Runtime/ld-linux-↩
   ↪x86-64.so.2
67 sgx.trusted_files.lnssdns = file:$(GRAPHENEDIR)/Runtime/↩
   ↪libnss_dns.so.2
68 # weitere Bibliotheken
69 sgx.trusted_files.lxml = file:/usr$(ARCH_LIBDIR)/libxml2.so.2
70 sgx.trusted_files.lssl = file:/usr$(ARCH_LIBDIR)/libssl.so↩
   ↪.1.1
71 sgx.trusted_files.lcrypto = file:/usr$(ARCH_LIBDIR)/libcrypto↩
   ↪.so.1.1
72 sgx.trusted_files.lz = file:$(ARCH_LIBDIR)/libz.so.1
73 sgx.trusted_files.lcurl = file:/usr$(ARCH_LIBDIR)/libcurl.so↩
   ↪.4
74 sgx.trusted_files.lpng = file:/usr$(ARCH_LIBDIR)/libpng16.so↩
   ↪.16
75 sgx.trusted_files.lonig = file:/usr$(ARCH_LIBDIR)/libonig.so↩
   ↪.4
76 sgx.trusted_files.lsqlite = file:/usr$(ARCH_LIBDIR)/↩
   ↪libsqlite3.so.0
```

```
77  sgx.trusted_files.licuuc = file:/usr$(ARCH_LIBDIR)/libicuuc.↩
      ↪so.60
78  sgx.trusted_files.llzma = file:$(ARCH_LIBDIR)/liblzma.so.5
79  sgx.trusted_files.lnghttp = file:/usr$(ARCH_LIBDIR)/↩
      ↪libnghttp2.so.14
80  sgx.trusted_files.lidn = file:/usr$(ARCH_LIBDIR)/libidn2.so.0
81  sgx.trusted_files.lrtmp = file:/usr$(ARCH_LIBDIR)/librtmp.so↩
      ↪.1
82  sgx.trusted_files.lpsl = file:/usr$(ARCH_LIBDIR)/libpsl.so.5
83  sgx.trusted_files.lssapikrb = file:/usr$(ARCH_LIBDIR)/↩
      ↪libgssapi_krb5.so.2
84  sgx.trusted_files.lldap = file:/usr$(ARCH_LIBDIR)/libldap_r↩
      ↪-2.4.so.2
85  sgx.trusted_files.llber = file:/usr$(ARCH_LIBDIR)/liblber↩
      ↪-2.4.so.2
86  sgx.trusted_files.licudata = file:/usr$(ARCH_LIBDIR)/↩
      ↪libicudata.so.60
87  sgx.trusted_files.lstdcxx = file:/usr$(ARCH_LIBDIR)/libstdc++↩
      ↪.so.6
88  sgx.trusted_files.lgcc = file:$(ARCH_LIBDIR)/libgcc_s.so.1
89  sgx.trusted_files.lunistring = file:/usr$(ARCH_LIBDIR)/↩
      ↪libunistring.so.2
90  sgx.trusted_files.lgnutls = file:/usr$(ARCH_LIBDIR)/libgnutls↩
      ↪.so.30
91  sgx.trusted_files.lhogweed = file:/usr$(ARCH_LIBDIR)/↩
      ↪libhogweed.so.4
92  sgx.trusted_files.lnettle = file:/usr$(ARCH_LIBDIR)/libnettle↩
      ↪.so.6
93  sgx.trusted_files.lgmp = file:/usr$(ARCH_LIBDIR)/libgmp.so.10
94  sgx.trusted_files.lkrb = file:/usr$(ARCH_LIBDIR)/libkrb5.so.3
95  sgx.trusted_files.lkcrypto = file:/usr$(ARCH_LIBDIR)/↩
      ↪libk5crypto.so.3
96  sgx.trusted_files.lcomerr = file:$(ARCH_LIBDIR)/libcom_err.so↩
      ↪.2
97  sgx.trusted_files.lkrbsupport = file:/usr$(ARCH_LIBDIR)/↩
      ↪libkrb5support.so.0
98  sgx.trusted_files.lsasl = file:/usr$(ARCH_LIBDIR)/libsasl2.so↩
      ↪.2
99  sgx.trusted_files.lssapi = file:/usr$(ARCH_LIBDIR)/libgssapi.↩
      ↪so.3
100 sgx.trusted_files.lpkit = file:/usr$(ARCH_LIBDIR)/libp11-kit.↩
```

```
    ↪so.0
101 sgx.trusted_files.ltasn = file:/usr$(ARCH_LIBDIR)/libtasn1.so↩
    ↪.6
102 sgx.trusted_files.lkeyutils = file:$(ARCH_LIBDIR)/libkeyutils↩
    ↪.so.1
103 sgx.trusted_files.lheimntlm = file:/usr$(ARCH_LIBDIR)/↩
    ↪libheimntlm.so.0
104 sgx.trusted_files.lkrbii = file:/usr$(ARCH_LIBDIR)/libkrb5.so↩
    ↪.26
105 sgx.trusted_files.lasn = file:/usr$(ARCH_LIBDIR)/libasn1.so.8
106 sgx.trusted_files.lhcrypto = file:/usr$(ARCH_LIBDIR)/↩
    ↪libhcrypto.so.4
107 sgx.trusted_files.lroken = file:/usr$(ARCH_LIBDIR)/libroken.↩
    ↪so.18
108 sgx.trusted_files.lffi = file:/usr$(ARCH_LIBDIR)/libffi.so.6
109 sgx.trusted_files.lwind = file:/usr$(ARCH_LIBDIR)/libwind.so↩
    ↪.0
110 sgx.trusted_files.lheimbase = file:/usr$(ARCH_LIBDIR)/↩
    ↪libheimbase.so.1
111 sgx.trusted_files.lhxcert = file:/usr$(ARCH_LIBDIR)/libhx509.↩
    ↪so.5
112 sgx.trusted_files.lcrypt = file:$(ARCH_LIBDIR)/libcrypt.so.1
113 sgx.trusted_files.libnsscompat = file:$(ARCH_LIBDIR)/↩
    ↪libnss_compat.so.2
114 sgx.trusted_files.libnssfiles  = file:$(ARCH_LIBDIR)/↩
    ↪libnss_files.so.2
115 sgx.trusted_files.lnssmdnsminimal = file:$(ARCH_LIBDIR)/↩
    ↪libnss_mdns4_minimal.so.2
116 sgx.trusted_files.lnssmyhostname = file:$(ARCH_LIBDIR)/↩
    ↪libnss_myhostname.so.2
117 sgx.trusted_files.lnsssystemd = file:$(ARCH_LIBDIR)/↩
    ↪libnss_systemd.so.2
118 sgx.trusted_files.lsodium = file:/usr$(ARCH_LIBDIR)/libsodium↩
    ↪.so.23
119 sgx.trusted_files.lzip = file:/usr$(ARCH_LIBDIR)/libzip.so.4
120 # Konfigurationsdateien
121 sgx.trusted_files.fpmconf = file:php-fpm.conf
122 sgx.trusted_files.wwwconf = file:php-fpm.d/www.conf
123 sgx.trusted_files.phpini = file:php.ini
124
125 # in der Enklave benötigte Dateien ohne Integritätscheck:
```

```
126 sgx.allowed_files.logdir = file:log
127 sgx.allowed_files.webroot = file:$(WEBROOT_DIR)
128 sgx.allowed_files.nsswitch = file:/etc/nsswitch.conf
129 sgx.allowed_files.passwd = file:/etc/passwd
130 sgx.allowed_files.group = file:/etc/group
131 sgx.allowed_files.hostconf = file:/etc/host.conf
132 sgx.allowed_files.resolvconf = file:/etc/resolv.conf
133 sgx.allowed_files.hosts = file:/etc/hosts
134 sgx.allowed_files.gaiconf = file:/etc/gai.conf
135 sgx.allowed_files.tmp = file:/tmp
```

Listing B.13: Inhalt der *Manifest*-Vorlage für PHP-FPM (Datei `php-fpm.manifest.template`)

```
1  ;
2  ; Konfigurationsdatei für den PHP FastCGI Process Manager
3  ;
4
5  [global]
6
7  ; Logdatei für Fehler
8  error_log = /var/log/php/fpm.log
9
10 ; Konfigurationsdateien für alle definierten Workerpools ↩
   ↪einbinden
11 include = php-fpm.d/*.conf
```

Listing B.14: Inhalt der PHP-FPM-Datei `php-fpm.conf`

```
1  ;
2  ; Konfigurationsdatei für den PHP-FPM Workerpool "www"
3  ;
4
5  [www]
6
7  ; der FPM kann mit Nutzerrechten ausgeführt werden
8  user  = elias
9  group = elias
10
11 ; akzeptiere FastCGI-Anfragen von allen IP-Adressen
12 ; auf Port 9000
```

```
listen = 9000

; Verhalten des Prozessmanagers festlegen:
; statisch: immer 4 Prozesse aktiv
; (dynamisch in Graphene-SGX nicht möglich)
pm = static
pm.max_children = 4
```

Listing B.15: Inhalt der PHP-FPM-Datei `php-fpm.d/www.conf`

```
;
; Konfigurationsdatei für jeden PHP FastCGI Prozess
;

[PHP]

; Fehlermeldungen nur in Logdatei schreiben
display_errors = Off
log_errors = On
error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT
error_log = /var/log/php/php.log

; Timeouts festlegen
max_execution_time = 60
max_input_time = 60

; akzeptiere große Uploads
post_max_size = 9G
upload_max_filesize = 9G

; weitere Einstellungen
output_buffering = 4096
register_argc_argv = Off
variables_order = "GPCS"
```

Listing B.16: Inhalt der PHP-FPM-Datei `php.ini`

Anhang C: Dateien für Performanztests

In diesem Kapitel wird der Inhalt aller Skript- und weiterer Dateien aufgeführt, welche für die in Abschnitt 4.2 beschriebenen Last- und Geschwindigkeitstests der migrierten WordPress-Instanz verwendet wurden.

C.1 Messungen mittels Siege

```
523   pthread_usleep_np(10000);
524
525   // Berechne die Zeit für eine komplette Anfrage
526   // (Download aller Webseiten-Dateien):
527   // Gesamtzeit / (parallele Nutzer * Wiederholungen pro ↩
       ↪Nutzer)
528   float request_time = data_get_total(data) / (my.cusers * my↩
       ↪.reps);
529
530   if (!my.quiet) {
531     if (my.failures > 0 && my.failed >= my.failures) {
532       fprintf(stderr, "%s␣aborted␣due␣to␣excessive␣socket␣↩
       ↪failure;␣you\n", program_name);
533       fprintf(stderr, "can␣change␣the␣failure␣threshold␣in␣↩
       ↪$HOME/.%src\n", program_name);
534     }
535     fprintf(stderr, "\nTransactions:\t\t%12u␣hits\n",        ↩
       ↪data_get_count(data));
536     fprintf(stderr, "Availability:\t\t%12.2f␣%%\n",         ↩
       ↪data_get_count(data)==0 ? 0 :
537                                                            ↩
       ↪(double)data_get_count(data) /
538                                                            ↩
       ↪(data_get_count(data)+my.failed)*100
539     );
```

```
540     fprintf(stderr, "Elapsed time:\t\t%12.2f secs\n",
    ↪ data_get_elapsed(data));
541     fprintf(stderr, "Data transferred:\t%12.2f MB\n",
    ↪ data_get_megabytes(data)); /*%12llu*/
542     fprintf(stderr, "Response time:\t\t%12.2f secs\n",
    ↪ data_get_response_time(data));
543     fprintf(stderr, "Request time:\t\t%12.6f secs\n",
    ↪ request_time); // Zeit fuer eine komplette Anfrage
544     fprintf(stderr, "Transaction rate:\t%12.2f trans/sec\n",
    ↪ data_get_transaction_rate(data));
545     fprintf(stderr, "Throughput:\t\t%12.2f MB/sec\n",
    ↪ data_get_throughput(data));
546     fprintf(stderr, "Concurrency:\t\t%12.2f\n",
    ↪ data_get_concurrency(data));
547     fprintf(stderr, "Successful transactions:%12u\n",
    ↪ data_get_code(data));
548     if (my.debug) {
549       fprintf(stderr, "HTTP OK received:\t%12u\n",
    ↪   data_get_okay(data));
550     }
551     fprintf(stderr, "Failed transactions:\t%12u\n",
    ↪ my.failed);
552     fprintf(stderr, "Longest transaction:\t%12.2f\n",
    ↪ data_get_highest(data));
553     fprintf(stderr, "Shortest transaction:\t%12.2f\n",
    ↪ data_get_lowest(data));
554     fprintf(stderr, " \n");
555   }
556
557   if (my.json_output) {
558     fprintf(stderr, "\n");
559     printf("{");
560     printf("\t\"transactions\":\t\t\t%12u,\n", data_get_count
    ↪ (data));
561
562     double availability;
563     if (data_get_count(data) == 0) {
564       availability = 0;
565     } else {
566       availability = (double)data_get_count(data) / (
    ↪ data_get_count(data) + my.failed) * 100;
```

```
567     }
568
569     printf("\t\"availability\":\t\t\t%12.2f,\n", availability↩
    ↪);
570     printf("\t\"elapsed_time\":\t\t\t%12.2f,\n", ↩
    ↪data_get_elapsed(data));
571     printf("\t\"data_transferred\":\t\t%12.2f,\n", ↩
    ↪data_get_megabytes(data)); /*%12llu*/
572     printf("\t\"response_time\":\t\t%12.2f,\n", ↩
    ↪data_get_response_time(data));
573     printf("\t\"request_time\":\t\t\t%12.6f,\n", request_time↩
    ↪); // Zeit fuer eine komplette Anfrage
574     printf("\t\"transaction_rate\":\t\t%12.2f,\n", ↩
    ↪data_get_transaction_rate(data));
575     printf("\t\"throughput\":\t\t\t%12.2f,\n", ↩
    ↪data_get_throughput(data));
576     printf("\t\"concurrency\":\t\t\t%12.2f,\n", ↩
    ↪data_get_concurrency(data));
577     printf("\t\"successful_transactions\":\t%12u,\n", ↩
    ↪data_get_code(data));
578
579     if (my.debug) {
580       printf("\t\"http_ok_received\":\t\t%12u,\n", ↩
    ↪data_get_okay(data));
581     }
582
583     printf("\t\"failed_transactions\":\t\t%12u,\n", my.failed↩
    ↪);
584     printf("\t\"longest_transaction\":\t\t%12.2f,\n", ↩
    ↪data_get_highest(data));
585     printf("\t\"shortest_transaction\":\t\t%12.2f\n", ↩
    ↪data_get_lowest(data));
586     puts("}");
587   }
```

Listing C.1: Ausschnitt aus der modifizierten Siege-Datei `siege/src/main.c`

```
1 #
2 # Konfigurationsdatei siegerc
3 #
4
```

```
verbose = false
color = on
quiet = false
logging = true
logfile = ./siege.log
# nicht nur HEAD-Anfragen
gmethod = GET
# nicht nur HTML-Datei herunterladen, sondern auch
# alle verlinkten CSS-, JavaScript- und weitere Dateien.
# => dadurch nicht nur eine Anfrage pro Wiederholung,
# sondern mehrere.
parser = true
# max. Threadanzahl
limit = 300
protocol = HTTP/1.1
chunked = true
# Cache deaktivieren
cache = false
# nicht keep-alive (sonst zu viele Server-Sessions)
connection = close
# bei jeder Anfrage eine neue Session aufbauen
expire-session = true
internet = false
accept-encoding = gzip, deflate
```

Listing C.2: Inhalt der Siege-Konfigurationsdatei `siegerc`

C.2 Messungen mittels cURL

C.2.1 Shellskripte

```
#!/bin/bash
TMP_FILE="media-download.tmp"

upload_year=$(date %+Y)
upload_month=$(date %+m)
results_file="results_media_download.csv"
filename_file="download-files.txt"
repeats=20

```

```
10 help()
11 {
12   echo "Optionen:"
13   echo -e "\t[-o␣<Ausgabedatei>]"
14   echo -e "\t[-f␣<Datei␣mit␣Dateinamen␣zum␣Download>]"
15   echo -e "\t[-r␣<Wiederholungen␣je␣Datei>]"
16   echo -e "\t[-y␣<Hochladejahr␣der␣Dateien>]"
17   echo -e "\t[-m␣<Hochlademonat␣der␣Dateien>]"
18   echo
19   echo "Standardwerte:"
20   echo -e "\to␣=␣$results_file"
21   echo -e "\tf␣=␣$filename_file"
22   echo -e "\tr␣=␣$repeats"
23   echo -e "\ty␣=␣$upload_year"
24   echo -e "\tm␣=␣$upload_month"
25 }
26
27 while getopts "o:f:r:y:m:h" opt; do
28   case $opt in
29     h)
30       help
31       exit 0
32       ;;
33     o)
34       results_file=$OPTARG
35       ;;
36     f)
37       filename_file=$OPTARG
38       ;;
39     r)
40       repeats=$OPTARG
41       ;;
42     y)
43       upload_year=$OPTARG
44       ;;
45     m)
46       upload_month=$OPTARG
47       ;;
48     ?)
49       echo "Fehler:␣Ungültige␣Option:␣$OPTARG."
50       echo
```

```
51        help
52        exit 1
53        ;;
54      :)
55        echo "Fehler: Argument fehlt für $OPTARG."
56        echo
57        help
58        exit 2
59        ;;
60    esac
61 done
62
63 url="https://localhost:8443/wp-content/uploads/$upload_year/$upload_month"
64
65 # Dezimaltrennung auf "." statt "," umstellen
66 # (bc kann nur mit "." rechnen)
67 LC_NUMERIC="en_US.UTF-8"
68
69 echo > $TMP_FILE # temporäre Datei anlegen
70
71 # Benchmark durchführen
72 for file in $(cat $filename_file); do
73   echo "Filename: $file"
74   for ((i=1; i <= $repeats; i++)); do
75     curl -s -k --write-out "%{speed_download};%{time_total};" --output /dev/null "$url/$file" >> $TMP_FILE
76     echo "$file" >> $TMP_FILE
77   done
78 done
79
80 echo "Berechne Mittelwerte..."
81 echo "filename;downloadSpeed;downloadTime" > $results_file
82 for file in $(cat $filename_file); do
83   totalSpeed=0
84   totalTime=0
85   # alle Werte aufsummieren
86   for line in $(grep $file $TMP_FILE); do # alle Zeilen für diese Datei
87     totalSpeed=$(echo "scale=3; $totalSpeed + $(echo $line | cut -f1 -d';')" | bc)
```

```
    totalTime=$(echo "scale=6; $totalTime + $(echo $line | cut -f2 -d';')" | bc)
  done
  # Mittelwerte berechnen
  totalSpeed=$(echo "scale=3; $totalSpeed / $repeats" | bc)
  totalTime=$(echo "scale=6; $totalTime / $repeats" | bc)
  # Ausgabe in Ergebnisdatei
  echo "$file;$totalSpeed;$totalTime" >> $results_file
done

rm $TMP_FILE # Aufräumen
```

Listing C.3: Inhalt des Shellskriptes `download-bench.sh`

```
#!/bin/bash
API_URL="https://localhost:8443/index.php?rest_route=/wp/v2"
CREDS="<Nutzername>:<Passwort>"
TMP_FILE="comment-upload.tmp"
TMP_FILE2="api-result.tmp"

results_file="results_comments.csv"
post_id=1
char_count=2
char_count_file=""
concurrent=1
repeats=5

# Dezimaltrennung auf "." statt "," umstellen
LC_NUMERIC="en_US.UTF-8"

help()
{
  echo "Optionen:"
  echo -e "\t[-o <Ausgabedatei>]"
  echo -e "\t[-p <Post-ID>]"
  echo -e "\t[-l <Zeichenlänge des Kommentars>]"
  echo -e "\t[-L <Datei mit versch. Zeichenlängen>] (überschreibt -l)"
  echo -e "\t[-n <Anzahl paralleler Nutzer>]"
  echo -e "\t[-r <Wiederholungen je Kommentar>]"
  echo
```

```
27    echo "Standard:"
28    echo -e "\to = $results_file"
29    echo -e "\tp = $post_id"
30    echo -e "\tl = $char_count"
31    echo -e "\tL = $char_count_file"
32    echo -e "\tn = $concurrent"
33    echo -e "\tr = $repeats"
34  }
35
36  add_float() # berechne $1 + $2
37  {
38    awk '{print ' $1 ' + ' $2 '}' <<< ""
39  }
40
41  divide_float() # berechne $1 / $2
42  {
43    awk '{print ' $1 ' / ' $2 '}' <<< ""
44  }
45
46  create_comment() # $1 = thread_id
47  {
48    tid=$1
49    sum=0
50    for ((i=1; i <= $repeats; i++)); do
51      # Kommentar veröffentlichen
52      curl_time=$(curl -k -s -X POST --user $CREDS -H "Content-↩
    ↪Type: application/json" --output "$TMP_FILE2.$tid" \
53          -d '{"author_name": "Kommentar '$(date +%s%N)'", "↩
    ↪author_email": "api@test.de", "content": "'$content'", "↩
    ↪post": '$post_id'}' \
54          --write-out "%{time_total}" $API_URL/comments)
55
56      result=$(jq ".id" $TMP_FILE2.$tid) # War Aktion ↩
    ↪erfolgreich?
57      if [[ $result == "null" ]]; then
58        echo "Fehler: Der $i. Kommentar in Thread #$tid konnte ↩
    ↪nicht erstellt werden."
59        echo "API-Response:"
60        cat $TMP_FILE2.$tid
61        exit 1
62      fi
```

```
63
64      sum=$(add_float $sum $curl_time)
65    done
66
67    echo -n "$count;" >> $TMP_FILE.$tid
68    divide_float $sum $repeats >> $TMP_FILE.$tid
69
70    echo -n "$tid␣"
71  }
72
73  while getopts "o:p:l:L:n:r:h" opt; do
74    case $opt in
75      h)
76        help
77        exit 0
78        ;;
79      o)
80        results_file=$OPTARG
81        ;;
82      p)
83        post_id=$OPTARG
84        if [[ $(curl -k -s $API_URL/posts/$post_id | jq ".id") ↩
    ↪== "null" ]]; then
85          echo "Fehler:␣Der␣Post␣#$post_id␣konnte␣nicht␣↩
    ↪gefunden␣werden."
86          echo "Bitte␣gib␣eine␣andere␣Post-ID␣an."
87          exit 1
88        fi
89        ;;
90      l)
91        char_count=$OPTARG
92        if [[ $char_count =~ ^-?[0-9]+$ ]]; then # Argumnet ↩
    ↪muss ein Integer sein
93          if [[ $char_count -lt 2 ]]; then
94            echo "Fehler:␣Der␣Kommentar␣muss␣mindestens␣2␣↩
    ↪Zeichen␣lang␣sein."
95            echo "Bitte␣gib␣eine␣Länge␣>=␣2␣an."
96            exit 1
97          fi
98        else
99          echo "Fehler:␣Ungültige␣Kommentarlänge."
```

```
100         echo "Bitte gib eine valide Zahl an."
101         exit 1
102       fi
103       ;;
104     L)
105       char_count_file=$OPTARG
106       ;;
107     n)
108       concurrent=$OPTARG
109       ;;
110
111     r)
112       repeats=$OPTARG
113       ;;
114     ?)
115       echo "Fehler: Ungültige Option: $OPTARG."
116       echo
117       help
118       exit 1
119       ;;
120     :)
121       echo "Fehler: Argument fehlt für Option $OPTARG."
122       echo
123       help
124       exit 2
125       ;;
126   esac
127 done
128
129 if [[ "$char_count_file." != "." ]]; then
130   if [[ ! -e $char_count_file ]]; then
131     echo "Fehler: Datei mit Zeichenlängen \"$char_count_file\"
      ↪" existiert nicht."
132     exit 1
133   fi
134   # Lies alle Längen aus der Datei ein
135   char_count=$(cat $char_count_file)
136 fi
137
138 # Gesamtzeit messen
139 start_time=$(date +%s)
```

```
140 for count in $char_count; do
141   echo -e "\nGeneriere zufälligen Kommentartext mit $count Zeichen..."
142   content=$(openssl rand -base64 $count | head -c $count | tr -d '\n')
143
144   printf "Starte Thread # "
145   for ((thread_id=1; thread_id <= $concurrent; thread_id++)); do
146     # Anfrage ausführen
147     create_comment $thread_id &
148     tids="$tids $!"
149     printf "%d " $thread_id
150   done
151   printf "\nBeendete Threads: # "
152
153   if [[ ".$tids." == ".." ]]; then
154     echo "Fehler: Thread IDs konnten nicht bestimmt werden."
155     exit 255
156   fi
157
158   wait $tids # warte, bis alle Threads fertig sind
159   echo
160 done
161 end_time=$(date +%s)
162
163 echo
164 echo "Uploads abgeschlossen. Berechne Mittelwerte..."
165
166 # Datei-Kopfzeile mit Testparametern schreiben
167 echo "INFO: concurrent=$concurrent, repeats=$repeats" > $results_file
168 echo "char_count;curl_time" >> $results_file
169 for count in $char_count; do
170   sum=0
171   for ((tid=1; tid <= $concurrent; tid++)); do
172     # aufsummieren
173     sum=$(add_float $sum $(grep $count $TMP_FILE.$tid | cut -f2 -d';'))
174   done
175   echo -n "$count;" >> $results_file
```

```
176   # Mittelwert bilden
177   divide_float $sum $concurrent >> $results_file
178 done
179
180 printf "Benötigte␣Gesamtzeit:␣%d,0␣s\n" $(($end_time - ↩
      ↪$start_time))
181 echo "Ergebnisse␣in␣Datei␣\"$results_file\"␣geschrieben."
182
183 # temporäre Dateien löschen
184 rm $TMP_FILE.*
185 rm $TMP_FILE2.*
186
187 exit 0
```

Listing C.4: Inhalt des Shellskriptes `comment-bench.sh`

C.2.2 Weitere Dateien

```
1  512b.txt
2  512k.txt
3  1m.txt
4  16m.txt
5  64m.txt
6  512m.txt
7  1g.txt
8  2g.txt
9  4g.txt
10 8g.txt
```

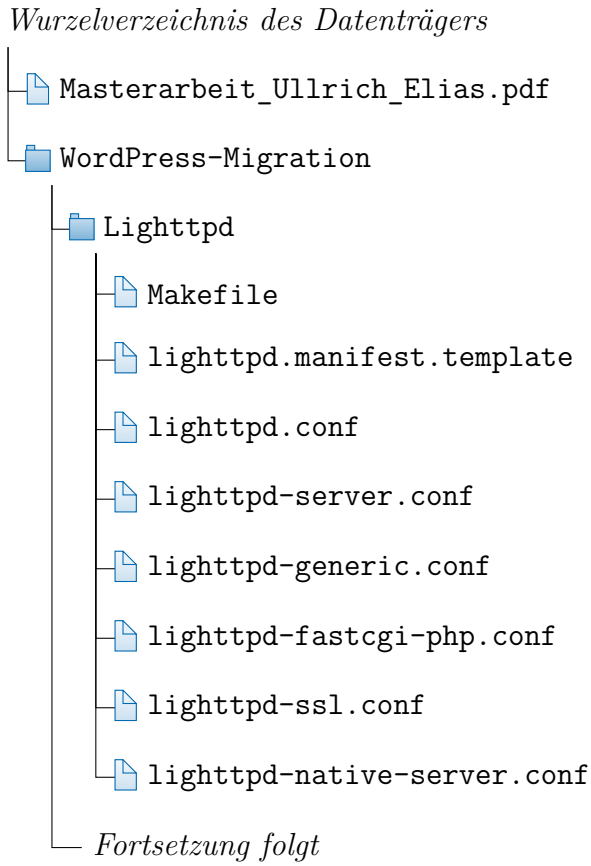
Listing C.5: Inhalt der Hilfsdatei `download-files.txt`

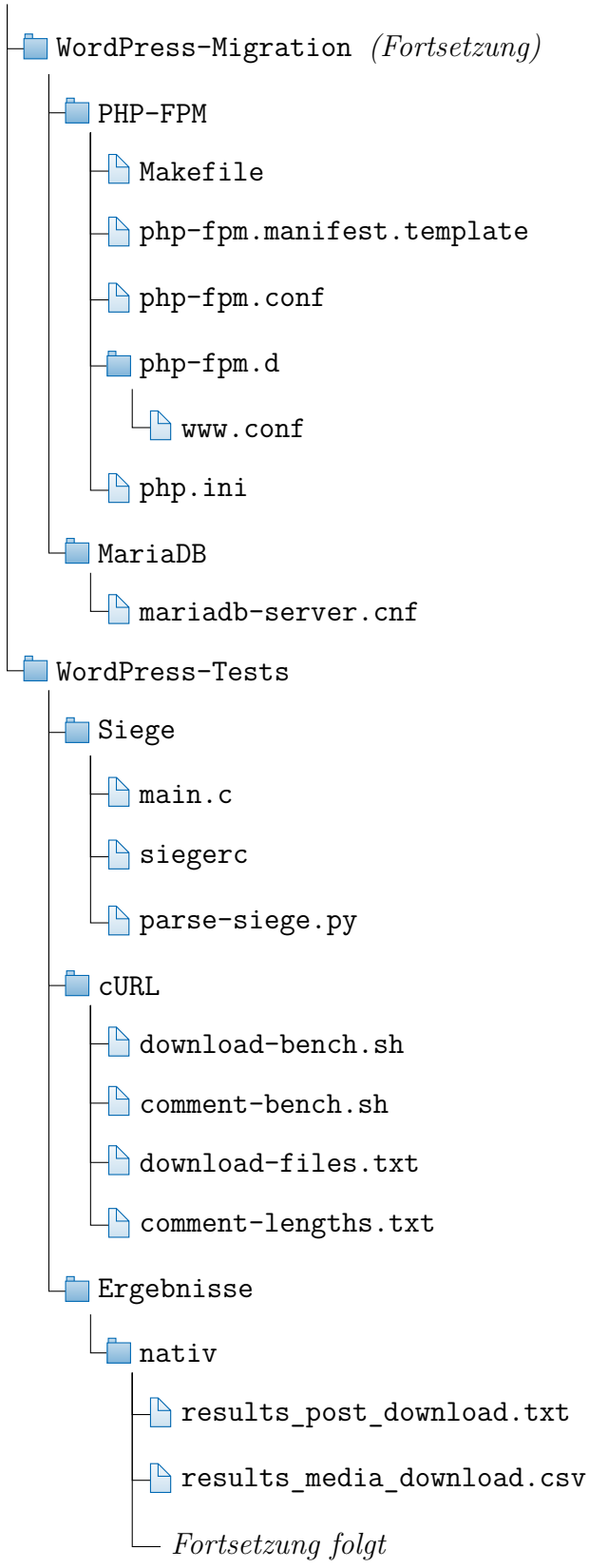
```
1 2
2 5000
3 30000
4 65525
```

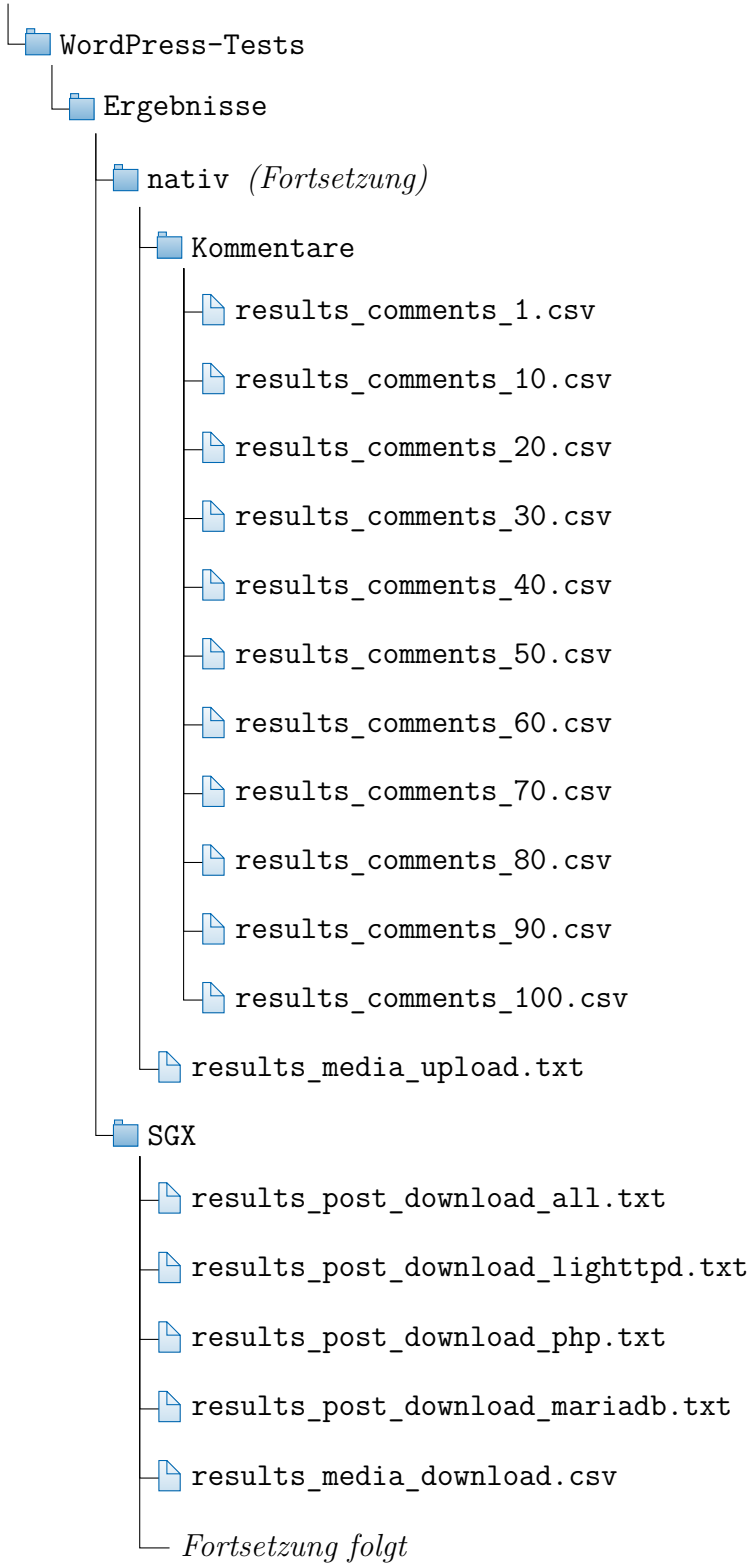
Listing C.6: Inhalt der Hilfsdatei `comment-lengths.txt`

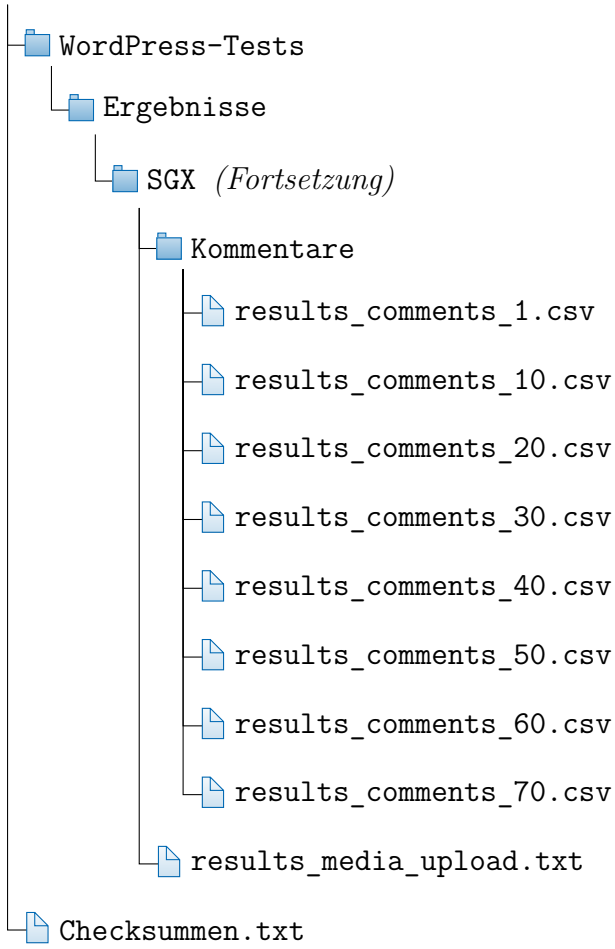
Anhang D: Inhalt des beiliegenden Datenträgers

Neben einer digitalen Version dieser Masterarbeit im PDF-Format werden alle in den vorhergehenden Kapiteln aufgeführten sowie weitere erwähnte Konfigurations-, Quelltext und andere Dateien sowie Shellskripte auf dem beiliegenden Datenträger bereitgestellt. Dessen Datei- und Verzeichnisstruktur wird im Folgenden aufgezeigt. Um die Integrität aller Inhalte sicherzustellen, werden die jeweiligen, mittels des SHA256-Algorithmus berechneten, Hashwerte in der Datei `Checksummen.txt` gespeichert.




WordPress-Migration (Fortsetzung)
PHP-FPM
Makefile
php-fpm.manifest.template
php-fpm.conf
php-fpm.d
www.conf
php.ini
MariaDB
mariadb-server.cnf
WordPress-Tests
Siege
main.c
siegerc
parse-siege.py
cURL
download-bench.sh
comment-bench.sh
download-files.txt
comment-lengths.txt
Ergebnisse
nativ
results_post_download.txt
results_media_download.csv
Fortsetzung folgt


WordPress-Tests
Ergebnisse
nativ (Fortsetzung)
Kommentare
results_comments_1.csv
results_comments_10.csv
results_comments_20.csv
results_comments_30.csv
results_comments_40.csv
results_comments_50.csv
results_comments_60.csv
results_comments_70.csv
results_comments_80.csv
results_comments_90.csv
results_comments_100.csv
results_media_upload.txt
SGX
results_post_download_all.txt
results_post_download_lighttpd.txt
results_post_download_php.txt
results_post_download_mariadb.txt
results_media_download.csv
Fortsetzung folgt


WordPress-Tests
Ergebnisse
SGX (Fortsetzung)
Kommentare
results_comments_1.csv
results_comments_10.csv
results_comments_20.csv
results_comments_30.csv
results_comments_40.csv
results_comments_50.csv
results_comments_60.csv
results_comments_70.csv
results_media_upload.txt
Checksummen.txt

Literaturverzeichnis

[Ada18a] Alexandre Adamski. *Overview of Intel SGX. Part 1, SGX Internals*. 5. Juli 2018. URL: https://blog.quarkslab.com/overview-of-intel-sgx-part-1-sgx-internals.html (besucht am 02. 06. 2020).

[Ada18b] Alexandre Adamski. *Overview of Intel SGX. Part 2 SGX Externals*. 2. Aug. 2018. URL: https://blog.quarkslab.com/overview-of-intel-sgx-part-2-sgx-externals.html (besucht am 03. 06. 2020).

[Alb08] Tom Alby. *Web 2.0. Konzepte, Anwendungen, Technologien*. 3., aktualisierte Auflage. München, Deutschland: Carl Hanser Verlag, 7. Feb. 2008. 229 S.

[Ana+13] Ittai Anati, Shay Gueron, Simon P. Johnson und Vincent R. Scarlata. *Innovative Technology for CPU Based Attestation and Sealing*. Techn. Whitepaper. Intel Corporation, 14. Aug. 2013. 7 S.

[Anj20] Anjuna Security, Inc., Hrsg. *Preventing Insider Threats With Anjuna Enterprise Enclaves™*. 2020. URL: https://www.anjuna.io/preventing-insider-threats-with-anjuna-enterprise-enclaves (besucht am 06. 11. 2020).

[Arn+16] Sergei Arnautov, Bohdan Trach, Franz Gregor et al. "SCONE: Secure Linux Containers with Intel SGX". In: *12th USENIX Symposium on Operating Systems Design and Implementation*. OSDI 2016. Savannah, Georgia, USA: USENIX Association, Nov. 2016, S. 689–703.

[Asy20] Asylo Authors, Hrsg. *Asylo. An open and flexible framework for enclave applications*. Version vom 30. Juli 2020. Google, Inc. 2020. URL: https://asylo.dev/ (besucht am 06. 11. 2020).

[Aut20] Automattic Inc., Hrsg. *Willkommen beim beliebtesten Website-Baukasten der Welt*. 2020. URL: https://de.wordpress.com/ (besucht am 29. 09. 2020).

[BPH15] Andrew Baumann, Marcus Peinado und Galen Hunt. "Shielding Applications from an Untrusted Cloud with Haven". In: *ACM Transactions on Computer Systems* 33.3, Art.-Nr. 8 (Aug. 2015). Hrsg. von Todd C. Mowry, S. 1–26.

[BL07] Ernie Brickell und Jiangtao Li. “Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities”. In: *IACR Cryptology ePrint Archive* 2007.194 (17. Aug. 2007), S. 1–36.

[Bro96] Mark R. Brown. *FastCGI Specification*. Hrsg. von Open Market, Inc. Version 1.0. 29. Apr. 1996. URL: `https://fastcgi-archives.github.io/FastCGI_Specification.html` (besucht am 01. 10. 2020).

[Car19] Eric Carter. *Sysdig 2019 Container Usage Report: New Kubernetes and security insights*. Sysdig, Inc. 29. Okt. 2019. URL: `https://sysdig.com/blog/sysdig-2019-container-usage-report/` (besucht am 22. 06. 2020).

[Che+08] Xiaoxin Chen, Tal Garfinkel, E. Christopher Lewis, Pratap Subrahmanyam, Carl A. Waldspurger, Dan Boneh, Jeffrey Dwoskin und Dan R. K. Ports. “Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems”. In: *SIGOPS Operating Systems Review* 42.2 (März 2008), S. 2–13.

[Chh+11] Siddhartha Chhabra, Brian Rogers, Yan Solihin und Milos Prvulovic. “SecureME: A Hardware-Software Approach to Full System Security”. In: *Proceedings of the International Conference on Supercomputing*. ICS 2011. New York City, New York, USA: Association for Computing Machinery, Mai 2011, S. 108–119.

[CMD16] Theo Combe, Antony Martin und Robert Di Pietro. “To Docker or Not to Docker: A Security Perspective”. In: *IEEE Cloud Computing* 3.5 (Okt. 2016), S. 54–62.

[Con20a] Confidential Computing Foundation, Hrsg. *Confidential Computing Consortium Projects*. 2020. URL: `https://confidentialcomputing.io/projects/` (besucht am 06. 11. 2020).

[Con20b] Confidential Computing Foundation, Hrsg. *Members*. 2020. URL: `https://confidentialcomputing.io/members/` (besucht am 06. 11. 2020).

[Con20c] Confidential Computing Foundation, Hrsg. *What is the Confidential Computing Consortium?* 2020. URL: `https://confidentialcomputing.io/` (besucht am 06. 11. 2020).

[Coo+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley und W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 3875. RFC Editor, Mai 2008. 36 S.

[CD16] Victor Costan und Srinivas Devadas. “Intel SGX Explained”. In: *IACR Cryptology ePrint Archive* 2016.86 (2016), S. 1–118.

[CLD16] Victor Costan, Ilia Lebedev und Srinivas Devadas. “Sanctum: Minimal Hardware Extensions for Strong Software Isolation”. In: *Proceedings of the 25th USENIX Security Symposium*. USENIX Security 2016. Austin, Texas, USA: USENIX Association, Aug. 2016, S. 857–874.

[CLD17] Victor Costan, Ilia Lebedev und Srinivas Devadas. “Secure Processors Part I. Background, Taxonomy for Secure Enclaves and Intel SGX Architecture”. In: *Foundations and Trends in Electronic Design Automation* 11.1–2 (2017), S. 1–248.

[CYS20] CYSEC SA, Hrsg. *Introducing the CYSEC's ARCA*. 2020. URL: `https://cysec.com/arca/` (besucht am 06. 11. 2020).

[DH76] Whitfield Diffie und Martin E. Hellman. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), S. 644–654.

[Doc20a] Docker Community, Hrsg. *Docker Official Image packaging for PHP. 7.4/buster/apache/Dockerfile*. Version vom 04. Nov. 2020. GitHub Repository. 2020. URL: `https://github.com/docker-library/php/blob/master/7.4/buster/apache/Dockerfile` (besucht am 10. 11. 2020).

[Doc20b] Docker Community, Hrsg. *Docker Official Image packaging for Redis. 6.0/Dockerfile*. Version vom 27. Okt. 2020. GitHub Repository. 2020. URL: `https://github.com/docker-library/redis/blob/master/6.0/Dockerfile` (besucht am 10. 11. 2020).

[Doc20c] Docker Community, Hrsg. *Docker Official Image packaging for WordPress*. Version vom 04. Sep. 2020. GitHub Repository. 2020. URL: `https://github.com/docker-library/wordpress` (besucht am 05. 10. 2020).

[Doc20d] Docker Community, Hrsg. *Docker Official Image packaging for WordPress. php7.4/apache/Dockerfile*. Version vom 30. Okt. 2020. GitHub Repository. 2020. URL: `https://github.com/docker-library/wordpress/blob/master/php7.4/apache/Dockerfile` (besucht am 10. 11. 2020).

[Doc20e] Docker, Inc., Hrsg. *Docker Desktop for Windows Stable Release notes. Stable Releases of 2016. Docker for Windows 1.12.0, 2016-07-28*. Docker Dokumentation. 2020. URL: `https://docs.docker.com/docker-for-windows/release-notes/#docker-for-windows-1120-2016-07-28` (besucht am 11. 11. 2020).

[Doc20f] Docker, Inc., Hrsg. *Dockerfile reference*. Docker Dokumentation. 2020. URL: `https://docs.docker.com/engine/reference/builder/` (besucht am 25. 06. 2020).

[Doc20g] Docker, Inc., Hrsg. *Install Docker Desktop on Windows Home*. Docker Dokumentation. 2020. URL: https://docs.docker.com/docker-for-windows/install-windows-home/ (besucht am 11. 11. 2020).

[Doc20h] Docker, Inc., Hrsg. *Overview of Docker Compose*. Docker Dokumentation. 2020. URL: https://docs.docker.com/compose/ (besucht am 10. 11. 2020).

[Doc20i] Docker, Inc., Hrsg. *wordpress*. Docker Hub. 2020. URL: https://hub.docker.com/_/wordpress (besucht am 10. 11. 2020).

[Eby10] P. J. Eby. *PEP 3333 -- Python Web Server Gateway Interface v1.0.1*. Hrsg. von Python Software Foundation. 4. Okt. 2010. URL: https://www.python.org/dev/peps/pep-3333/ (besucht am 01. 10. 2020).

[Fel+15] Wes Felter, Alexandre Ferreira, Ram Rajamony und Juan Rubio. "An Updated Performance Comparison of Virtual Machines and Linux Containers". In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software*. ISPASS 2015. Philadelphia, Pennsylvania, USA: The Institute of Electrical und Electronics Engineers, Inc., März 2015, S. 171–172.

[For20] Fortanix Inc., Hrsg. *Fortanix. Data-First Multicloud Security*. 2020. URL: https://www.fortanix.com/ (besucht am 06. 11. 2020).

[Fre20] Free Software Foundation, Inc., Hrsg. *The GNU C Library (glibc). What is glibc?* 2020. URL: https://www.gnu.org/software/libc/ (besucht am 12. 08. 2020).

[Ful12] Jeff Fulmer. *Siege Home*. 30. Jan. 2012. URL: https://www.joedog.org/siege-home/ (besucht am 22. 10. 2020).

[Ful20] Jeff Fulmer. *Siege*. Version vom 15. Jul. 2020. GitHub Repository. 2020. URL: https://github.com/JoeDog/siege (besucht am 22. 10. 2020).

[Gen09] Craig Gentry. "A Fully Homomorphic Encryption Scheme". Diss. Stanford, California, USA: Stanford University, Sep. 2009. 209 S.

[Geo19a] Georgia Institute of Technology (Systems Software & Security Lab), Hrsg. *SGX Bootstrap. Overview*. Juli 2019. URL: https://sgx101.gitbook.io/sgx101/sgx-bootstrap/overview (besucht am 09. 07. 2020).

[Geo19b] Georgia Institute of Technology (Systems Software & Security Lab), Hrsg. *SGX Bootstrap. Enclave*. Juli 2019. URL: https://sgx101.gitbook.io/sgx101/sgx-bootstrap/enclave (besucht am 10. 06. 2020).

[Geo19c] Georgia Institute of Technology (Systems Software & Security Lab), Hrsg. *SGX Bootstrap. Attestation*. Juli 2019. URL: https://sgx101.gitbook.io/sgx101/sgx-bootstrap/attestation (besucht am 30. 07. 2020).

[Geo19d] Georgia Institute of Technology (Systems Software & Security Lab), Hrsg. *SGX Bootstrap. Communication between Architectural and Application Enclaves*. Juli 2019. URL: https://sgx101.gitbook.io/sgx101/sgx-bootstrap/enclave/interaction-between-pse-and-application-enclaves (besucht am 28. 07. 2020).

[Git20a] Vivek Gite. *How to set up MariaDB SSL and secure connections from clients*. Hrsg. von nixCraft. Version vom 06. Nov. 2020. 2020. URL: https://www.cyberciti.biz/faq/how-to-setup-mariadb-ssl-and-secure-connections-from-clients/ (besucht am 12. 11. 2020).

[Git20b] GitHub, Inc., Hrsg. *Graphene Library OS with Intel SGX Support. Commit Activity*. Version vom 03. Okt. 2020. GitHub Repository. 2020. URL: https://github.com/oscarlab/graphene/graphs/commit-activity (besucht am 09. 10. 2020).

[Gra19a] Graphene Contributors, Hrsg. *Building*. Graphene Dokumentation. 2019. URL: https://graphene.readthedocs.io/en/latest/building.html (besucht am 09. 10. 2020).

[Gra19b] Graphene Contributors, Hrsg. *gsc – Graphene Shielded Containers*. Graphene Dokumentation. 2019. URL: https://graphene.readthedocs.io/en/latest/manpages/gsc.html (besucht am 10. 11. 2020).

[Gra19c] Graphene Contributors, Hrsg. *Introduction to Graphene*. Graphene Dokumentation. 2019. URL: https://graphene.readthedocs.io/en/latest/index.html (besucht am 09. 11. 2020).

[Gra19d] Graphene Contributors, Hrsg. *Manifest syntax*. Graphene Dokumentation. 2019. URL: https://graphene.readthedocs.io/en/latest/manifest-syntax.html (besucht am 26. 11. 2020).

[Gra19e] Graphene Contributors, Hrsg. *Performance tuning and analysis. Exitless Feature*. Graphene Dokumentation. 2019. URL: https://graphene.readthedocs.io/en/latest/perf-practices.html#exitless-feature (besucht am 04. 11. 2020).

[gst20] gstrauss. *Setting up a simple SSL configuration with a self-signed certificate*. Hrsg. von Jean-Philippe Lang. Version vom 28. Mai 2020. Lighttpd Dokumentation. 2020. URL: https://redmine.lighttpd.net/projects/lighttpd/wiki/HowToSimpleSSL (besucht am 14. 10. 2020).

[HM08] Serge E. Hallyn und Andrew G. Morgan. “Linux Capabilities: making them work”. In: *Proceedings of the Linux Symposium*. Bd. 1. Ottawa, Ontario, Kanada, Juli 2008, S. 163–172.

[Hax20] Haxx, Hrsg. *curl://. command line tool and library for transferring data with URLs*. 2020. URL: `https://curl.haxx.se/` (besucht am 22. 10. 2020).

[Hei+20] Peter Heidkamp, Marko Vogel, Axel Pols und Lukas Gentemann. *Cloud-Monitor 2020. Die Integrationsfähigkeit und Interoperabilität der Cloud stärken*. Techn. Ber. Köln, Deutschland: KPMG AG Wirtschaftsprüfungsgesellschaft und Bitkom Research GmbH, 2020. 37 S.

[HBB18] Kelsey Hightower, Brendan Burns und Joe Beda. *Kubernetes. Eine kompakte Einführung*. Englisch übers. von Thomas Demmig. Heidelberg: dpunkt.verlag, 2018. 204 S.

[Hoe13] Matthew E. Hoekstra. *Intel® SGX for Dummies (Intel® SGX Design Objectives)*. Hrsg. von Intel Corporation. 26. Sep. 2013. URL: `https://software.intel.com/content/www/us/en/develop/blogs/protecting-application-secrets-with-intel-sgx.html` (besucht am 09. 07. 2020).

[Inc20] Inclavare Containers, Hrsg. *Inclavare Containers*. 2020. URL: `https://inclavare-containers.io/en/` (besucht am 06. 11. 2020).

[Int17] Intel Corporation, Hrsg. *Getting Started with Intel® Software Guard Extensions SDK for Microsoft* Windows* OS*. 21. Juni 2017. URL: `https://software.intel.com/content/www/us/en/develop/articles/getting-started-with-sgx-sdk-for-windows.html` (besucht am 28. 07. 2020).

[Int20a] Intel Corporation, Hrsg. *10th Generation Intel® Core™ Processor Families. Datasheet, Volume 1 of 2*. Version 005. Juli 2020. 128 S.

[Int20b] Intel Corporation, Hrsg. *Build an Intel® Software Guard Extensions ECDSA Attestation Service to Strengthen Enclave Security. Intel® Provisioning Certification Service for ECDSA Attestation*. 2020. URL: `https://api.portal.trustedservices.intel.com/provisioning-certification` (besucht am 03. 08. 2020).

[Int20c] Intel Corporation, Hrsg. *Intel Xeon Scalable Platform Built for Most Sensitive Workloads*. 14. Okt. 2020. URL: `https://newsroom.intel.com/news-releases/intel-xeon-scalable-platform-built-most-sensitive-workloads/` (besucht am 17. 11. 2020).

[Int20d] Intel Corporation, Hrsg. *Intel(R) Software Guard Extensions Data Center Attestation Primitives*. Version vom 04. Sep. 2020. GitHub Repository. 2020. URL: `https://github.com/intel/SGXDataCenterAttestationPrimitives` (besucht am 03. 08. 2020).

[Int20e] Intel Corporation, Hrsg. *Intel(R) Software Guard Extensions for Linux* OS. linux-sgx-driver*. Version vom 30. Mai 2020. GitHub Repository. 2020. URL: `https://github.com/intel/linux-sgx-driver` (besucht am 28. 07. 2020).

[Int20f] Intel Corporation, Hrsg. *Intel(R) Software Guard Extensions for Linux* OS. linux-sgx*. Version vom 07. Jul. 2020. GitHub Repository. 2020. URL: `https://github.com/intel/linux-sgx` (besucht am 28. 07. 2020).

[Int20g] Intel Corporation, Hrsg. *Intel(R) Software Guard Extensions for Linux* OS. Intel(R) SGX Reference Launch Enclave*. Version vom 06. Jul. 2020. GitHub Repository. 2020. URL: `https://github.com/intel/linux-sgx/blob/master/psw/ae/ref_le/ref_le.md` (besucht am 03. 08. 2020).

[Int20h] Intel Corporation, Hrsg. *Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3D: System Programming Guide, Part 4*. Ten-Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals. Version 072. Santa Clara, California, USA: Intel Corporation, Mai 2020. 274 S.

[Int20i] Intel Corporation, Hrsg. *Intel® Software Guard Extensions (Intel® SGX). Developer Guide*. Version 2.9.1. Santa Clara, California, USA: Intel Corporation, Apr. 2020. 52 S.

[Int20j] Intel Corporation, Hrsg. *Intel® Software Guard Extensions (Intel® SGX). Was ist Intel® SGX?* 2020. URL: `https://www.intel.de/content/www/de/de/architecture-and-technology/software-guard-extensions.html` (besucht am 02. 11. 2020).

[Int20k] Intel Corporation, Hrsg. *Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS. Developer Reference*. Version 2.9.1. Santa Clara, California, USA: Intel Corporation, Apr. 2020. 405 S.

[Int20l] Intel Corporation, Hrsg. *Intel® Software Guard Extensions (Intel® SGX) SDK for Windows* OS. Developer Reference*. Version 2.7. Santa Clara, California, USA: Intel Corporation, Mai 2020. 461 S.

[Int20m] Intel Corporation, Hrsg. *Produkte mit früherer Bezeichnung Comet Lake*. Intel Produktübersicht. 2020. URL: https://ark.intel.com/content/www/de/de/ark/products/codename/90354/comet-lake.html (besucht am 11. 08. 2020).

[ISO16] ISO. *Information technology – Reference Architecture for Service Oriented Architecture (SOA RA). Part 1: Terminology and Concepts for SOA*. ISO/IEC 18384-1:2016(E). 1. Edition. Vernier, Schweiz: International Organization for Standardization, 1. Juni 2016. 51 S.

[Joh+16] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell und Frank Mckeen. *Intel® SGX: Intel® EPID Provisioning and Attestation Services*. Techn. Whitepaper. Intel Corporation, 9. März 2016. 10 S.

[Joh18] Simon Paul Johnson. *An update on 3rd Party Attestation*. Hrsg. von Intel Corporation. 9. Dez. 2018. URL: https://software.intel.com/content/www/us/en/develop/blogs/an-update-on-3rd-party-attestation.html (besucht am 12. 06. 2020).

[JBZ16] Simon Paul Johnson, Derek Bombien und Dan T. Zimmerman. *Intel® SGX: Debug, Production, Pre-release – What's the Difference?* Intel Corporation. 8. Jan. 2016. URL: https://software.intel.com/content/www/us/en/develop/blogs/intel-sgx-debug-production-prelease-whats-the-difference.html (besucht am 02. 11. 2020).

[KM19] Sean P. Kane und Karl Matthias. *Docker: Up & Running. Shipping Reliable Containers in Production*. Hrsg. von Virginia Wilson, Justin Billing und Rachel Monaghan. 2. Aufl. 2. Release. Sebastopol, California, USA: O'Reilly Media, Inc., 22. März 2019. 326 S.

[KPW16] David Kaplan, Jeremy Powell und Tom Woller. *AMD Memory Encryption*. Techn. Whitepaper. Advanced Micro Devices, Inc., 21. Apr. 2016. 12 S.

[KK08] Andi Kleen und Michael Kerrisk. *tcp(7) — Linux manual page*. Version 2020-06-09. 2008. URL: https://man7.org/linux/man-pages/man7/tcp.7.html (besucht am 16. 09. 2020).

[KKS16] Andi Kleen, Michael Kerrisk und Heinrich Schuchardt. *unix(7) — Linux manual page*. Version 2020-06-09. 2016. URL: https://man7.org/linux/man-pages/man7/unix.7.html (besucht am 17. 09. 2020).

[Kör12] Helmut Körner. *DevOps. Praktische Anleitung und Vorgaben*. V 1.0. Morisville, North Carolina, USA: Lulu Press, Inc., 1. Okt. 2012. 127 S.

[Lan20] Jean-Philippe Lang, Hrsg. *Lighttpd*. 2020. URL: https://redmine.lighttpd.net/projects/lighttpd (besucht am 05. 10. 2020).

[Lin+17] Joshua Lind, Christian Priebe, Divya Muthukumaran et al. “Glamdring: Automatic Application Partitioning for Intel SGX”. In: *2017 USENIX Annual Technical Conference*. USENIX ATC 2017. Santa Clara, California, USA: USENIX Association, Juli 2017, S. 285–298.

[LHS05] Xue Liu, Jin Heo und Lui Sha. “Modeling 3-tiered Web applications”. In: *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. MASCOTS ’05. Atlanta, Georgia, USA: The Institute of Electrical und Electronics Engineers, Inc., Sep. 2005, S. 307–310.

[MAD20] MADANA UG, Hrsg. *MADANA Core*. 2020. URL: `https://www.madana.io/core.html` (besucht am 06. 11. 2020).

[Mar20a] MariaDB Foundation, Hrsg. *Client/Server Protocol*. MariaDB Dokumentation. 2020. URL: `https://mariadb.com/kb/en/clientserver-protocol/` (besucht am 07. 10. 2020).

[Mar20b] MariaDB Foundation, Hrsg. *Connecting to MariaDB. protocol*. MariaDB Dokumentation. 2020. URL: `https://mariadb.com/kb/en/connecting-to-mariadb/#protocol` (besucht am 07. 10. 2020).

[Mar20c] MariaDB Foundation, Hrsg. *MariaDB Server: The open source relational database*. 2020. URL: `https://mariadb.org/` (besucht am 30. 09. 2020).

[Mar20d] MariaDB Foundation, Hrsg. *Securing Connections for Client and Server*. MariaDB Dokumentation. 2020. URL: `https://mariadb.com/kb/en/securing-connections-for-client-and-server/` (besucht am 07. 10. 2020).

[McC20] Kylie McClain, Hrsg. *musl libc. a new libc striving to be fast, simple, lightweight, free and correct*. Community Wiki. 2020. URL: `https://wiki.musl-libc.org/` (besucht am 14. 08. 2020).

[McK+16] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd und Carlos Rozas. “Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave”. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. HASP 2016 10. Intel Corporation. New York City, New York, USA: Association for Computing Machinery, Juni 2016, S. 1–9.

[McK13] Siobhan McKeown. *WordPress. Freedom, Community and the Business of Open Source*. WordPress.org, 31. Mai 2013. Kap. 3: On Forking WordPress, Forks in General, Early WordPress and the Community. 36 S.

[Mec16] John P. Mechalas. *Intel® Software Guard Extensions Part 7: Refine the Enclave with Proxy Functions*. Hrsg. von Intel Corporation. 28. Nov. 2016. URL: https://software.intel.com/content/www/us/en/develop/articles/intel-software-guard-extensions-tutorial-part-7-refining-the-enclave.html (besucht am 27. 07. 2020).

[Mec17] John P. Mechalas. *Properly Detecting Intel® Software Guard Extensions (Intel® SGX) in Your Applications*. Hrsg. von Intel Corporation. 8. Aug. 2017. URL: https://software.intel.com/content/www/us/en/develop/articles/properly-detecting-intel-software-guard-extensions-in-your-applications.html (besucht am 28. 07. 2020).

[MO16] John P. Mechalas und Benjamin J. Odom. *Intel® Software Guard Extensions Tutorial Series: Part 1, Intel® SGX Foundation*. Hrsg. von Intel Corporation. 7. Juli 2016. URL: https://software.intel.com/content/www/us/en/develop/articles/intel-software-guard-extensions-tutorial-part-1-foundation.html (besucht am 27. 07. 2020).

[MR16] John P. Mechalas und Isayah Reed. *Intel® Software Guard Extensions Tutorial Series: Part 3, Design an Application*. Hrsg. von Intel Corporation. 29. Aug. 2016. URL: https://software.intel.com/content/www/us/en/develop/articles/software-guard-extensions-tutorial-series-part-3.html (besucht am 27. 07. 2020).

[MG11] Peter Mell und Timothy Grance. *The NIST Definition of Cloud Computing*. NIST Special Publication (SP) 800-145. Gaithersburg, Maryland, USA: National Institute of Standards und Technology, Sep. 2011. 7 S.

[Mic20] Microsoft Corporation, Hrsg. *Confidential computing nodes on Azure Kubernetes Service (public preview). AKS Provided Daemon Sets (addon)*. Azure Dokumentation. 22. Sep. 2020. URL: https://docs.microsoft.com/en-us/azure/confidential-computing/confidential-nodes-aks-overview#aks-provided-daemon-sets-addon (besucht am 26. 11. 2020).

[Min18] Marina Minkin. "Improving Performance and Security of Intel SGX". Masterarbeit. Haifa, Israel: Technion – Israel Institute of Technology, Dez. 2018. 62 S.

[Mof+18] Saeid Mofrad, Fengwei Zhang, Shiyong Lu und Weidong Shi. "A Comparison Study of Intel SGX and AMD Memory Encryption Technology". In: *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP 2018. New York City, New York, USA: Association for Computing Machinery, 2. Juni 2018, S. 1–8.

[Nad+16] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty und Mike Amundsen. *Microservice Architecture. Aligning Principles, Practices, and Culture*. Hrsg. von Brian MacDonald, Holly Bauer, Kristen Brown und Christina Edwards. 1. Aufl. 2. Release. Sebastopol, California, USA: O'Reilly Media, Inc., 18. Juli 2016. 127 S.

[NLV11] Michael Naehrig, Kristin Lauter und Vinod Vaikuntanathan. "Can Homomorphic Encryption be Practical?" In: *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. CCSW 2011. New York City, New York, USA: Association for Computing Machinery, 21. Okt. 2011, S. 113–124.

[Nam19] Daye Nam. "API Design Implications of Boilerplate Client Code". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering*. ASE 2019. San Diego, California, USA: The Institute of Electrical und Electronics Engineers, Inc., Nov. 2019, S. 1253–1255.

[Nga+16] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho und Sarah Martin. "TrustZone Explained: Architectural Features and Use Cases". In: *2016 IEEE 2nd International Conference on Collaboration and Internet Computing*. IEEE CIC 2016. Pittsburgh, Pennsylvania, USA: The Institute of Electrical und Electronics Engineers, Inc., Nov. 2016, S. 445–451.

[Occ20] Occlum team, Hrsg. *Occlum. Empowering everyone to run every app in enclaves*. Version vom 10. Nov. 2020. GitHub Repository. 2020. URL: `https://github.com/occlum/occlum` (besucht am 10. 11. 2020).

[Ora20] Oracle Corporation, Hrsg. *Oracle Database 19c*. 2020. URL: `https://www.oracle.com/de/database/technologies/` (besucht am 30. 09. 2020).

[Ore+17] Meni Orenbach, Pavel Lifshits, Marina Minkin und Mark Silberstein. "Eleos: ExitLess OS Services for SGX Enclaves". In: *Proceedings of the Twelfth European Conference on Computer Systems*. EuroSys 2017. New York City, New York, USA: Association for Computing Machinery, Apr. 2017, S. 238–253.

[OSC20a] OSCAR Lab, Hrsg. *Graphene Library OS with Intel SGX Support. SGX Tools. RA-TLS Libraries*. Version vom 28. Sep. 2020. GitHub Repository. 2020. URL: `https://github.com/oscarlab/graphene/tree/master/Pal/src/host/Linux-SGX/tools#ra-tls-libraries` (besucht am 03. 11. 2020).

[OSC20b] OSCAR Lab, Hrsg. *Graphene Library OS with Intel SGX Support. Secret Provisioning Minimal Examples.* Version vom 22. Okt. 2020. GitHub Repository. 2020. URL: https://github.com/oscarlab/graphene/tree/master/Examples/ra-tls-secret-prov (besucht am 03. 11. 2020).

[OSC20c] OSCAR Lab, Hrsg. *Graphene Library OS with Intel SGX Support. Protected Files.* Version vom 22. Okt. 2020. GitHub Repository. 2020. URL: https://github.com/oscarlab/graphene/tree/master/Pal/src/host/Linux-SGX/protected-files (besucht am 04. 11. 2020).

[OSC20d] OSCAR Lab, Hrsg. *Graphene Library OS with Intel SGX Support. SGX Tools. Secret Provisioning Libraries.* Version vom 22. Okt. 2020. GitHub Repository. 2020. URL: https://github.com/oscarlab/graphene/tree/master/Pal/src/host/Linux-SGX/tools#secret-provisioning-libraries (besucht am 04. 11. 2020).

[OSC20e] OSCAR Lab, Hrsg. *Graphene Library OS with Intel SGX Support. Commits.* Version vom 09. Okt. 2020. GitHub Repository. 2020. URL: https://github.com/oscarlab/graphene/commits/master (besucht am 10. 11. 2020).

[OSC20f] OSCAR Lab, Hrsg. *Graphene SGX Driver.* Version vom 02. Nov. 2020. GitHub Repository. 2020. URL: https://github.com/oscarlab/graphene-sgx-driver (besucht am 04. 11. 2020).

[Pri+20] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, Vasily A. Sartakov und Peter Pietzuch. "SGX-LKL: Securing the Host OS Interface for Trusted Execution". In: *arXiv e-prints*, Art.-Nr. arXiv:1908.11143v3 [cs.OS] (20. Jan. 2020), S. 1–17.

[pro20] procps-ng, Hrsg. *pmap(1) — Linux manual page.* Version 2020-06-04. 2020. URL: https://www.man7.org/linux/man-pages/man1/pmap.1.html (besucht am 04. 11. 2020).

[PGT10] Octavian Purdila, Lucian Adrian Grijincu und Nicolae Tapus. "LKL: The Linux kernel library". In: *9th RoEduNet IEEE International Conference.* Sibiu, Rumänien: The Institute of Electrical und Electronics Engineers, Inc., Juni 2010, S. 328–333.

[QSu20] Q-Success, Hrsg. *Usage statistics of content management systems.* 2020. URL: https://w3techs.com/technologies/overview/content_management (besucht am 05. 10. 2020).

[Qui+17] Daniel Quinlan, Michael Kerrisk, Alan Cox, Michael Neuffer und Andries Brouwer. *proc(5) — Linux manual page*. Version 2020-08-13. 2017. URL: https://man7.org/linux/man-pages/man5/proc.5.html (besucht am 06. 10. 2020).

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. RFC Editor, Aug. 2018. 160 S.

[RQK15] Adam J. Richter, Daniel Quinlan und Michael Kerrisk. *dlopen(3) — Linux manual page*. Version 2020-06-09. 2015. URL: https://man7.org/linux/man-pages/man3/dlopen.3.html (besucht am 16. 09. 2020).

[RC04] D. Robinson und K. Coar. *The Common Gateway Interface (CGI) Version 1.1*. RFC 5280. RFC Editor, Okt. 2004. 151 S.

[Rus18] Mark Russinovich. *Azure confidential computing*. Hrsg. von Microsoft Corporation. 9. Mai 2018. URL: https://azure.microsoft.com/de-de/blog/azure-confidential-computing/ (besucht am 06. 11. 2020).

[SAB15] Mohamed Sabt, Mohammed Achemlal und Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not". In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Bd. 1. TrustCom 2015. Helsinki, Finnland: The Institute of Electrical und Electronics Engineers, Inc., Aug. 2015, S. 57–64.

[Sca+19] Vinnie Scarlata, Simon Johnson, James Beaney und Piotr Zmijewski. *Supporting Third Party Attestation for Intel® SGX with Intel® Data Center Attestation Primitives*. Techn. Whitepaper. Intel Corporation, 19. Apr. 2019. 8 S.

[Sch08] Neil Schemenauer. *SCGI: A Simple Common Gateway Interface alternative*. 23. Juni 2008. URL: https://python.ca/scgi/protocol.txt (besucht am 01. 10. 2020).

[Sch20] Felix Schuster. *Edgeless Systems. Products*. Hrsg. von Edgeless Systems GmbH. 2020. URL: https://www.edgeless.systems/products (besucht am 06. 11. 2020).

[SCO20a] SCONTAIN UG, Hrsg. *Running "Hello World" inside of an enclave*. Version vom 08. Okt. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/hardwaremode/ (besucht am 09. 10. 2020).

[SCO20b] SCONTAIN UG, Hrsg. *SCONE CAS Concepts*. Version vom 02. Nov. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/cas_intro/ (besucht am 03. 11. 2020).

[SCO20c] SCONTAIN UG, Hrsg. *SCONE Configuration and Attestation Service (CAS)*. Version vom 02. Nov. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/CASOverview/ (besucht am 03. 11. 2020).

[SCO20d] SCONTAIN UG, Hrsg. *SCONE Curated Images*. Version vom 08. Okt. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/curated_images/ (besucht am 09. 10. 2020).

[SCO20e] SCONTAIN UG, Hrsg. *SCONE Environment Variables*. Version vom 08. Okt. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/SCONE_ENV/ (besucht am 09. 10. 2020).

[SCO20f] SCONTAIN UG, Hrsg. *SCONE File Protection. Volumes*. Version vom 19. Okt. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/SCONE_Fileshield/ (besucht am 21. 10. 2020).

[SCO20g] SCONTAIN UG, Hrsg. *SCONE Session Language (v0.2)*. Version vom 19. Okt. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/CAS_session_lang_0_2/ (besucht am 21. 10. 2020).

[SCO20h] SCONTAIN UG, Hrsg. *Sconify Container Images (Community Version)*. Version vom 09. Nov. 2020. SCONE Dokumentation. 2020. URL: https://sconedocs.github.io/sconify_image/ (besucht am 10. 11. 2020).

[She+20] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia und Shoumeng Yan. "Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS 2020. New York City, New York, USA: Association for Computing Machinery, März 2020, S. 955–970.

[Shi+17] Shweta Shinde, Dat Le Tien, Shruti Tople und Prateek Saxena. "Panoply: Low-TCB Linux Applications With SGX Enclaves". In: *NDSS Symposium 2017*. NDSS '17. San Diego, California, USA: Internet Society, Feb. 2017, S. 1–15.

[SH99] P. Srisuresh und M. Holdrege. *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663. RFC Editor, Aug. 1999. 30 S.

[SP08] Simon Stobart und David Parsons. *Dynamic Web Application Development: Using PHP and MySQL*. Hrsg. von Gaynor Redvers-Mutton und Lucy Mills. London, England: Cengage Learning EMEA, 2008. 440 S.

[Sul20] Denise Sullivan. *The Risks and Benefits of Cloud Storage*. Hrsg. von Cloudwards.net. 6. Juli 2020. URL: https://www.cloudwards.net/the-risks-and-benefits-of-cloud-storage/ (besucht am 30. 10. 2020).

[Syn20] Synopsys, Inc., Hrsg. *The Heartbleed Bug*. Version vom 03. Juni 2020. 2020. URL: https://heartbleed.com/ (besucht am 02. 11. 2020).

[Syv94] Paul Syverson. "A taxonomy of replay attacks". In: *Proceedings The Computer Security Foundations Workshop VII*. Franconia, New Hampshire, USA: The Institute of Electrical und Electronics Engineers, Inc., Juni 1994, S. 187–191.

[Tan09] Andrew S. Tanenbaum. *Moderne Betriebssysteme*. 3., aktualisierte Aufl. München: Pearson Education Deutschland GmbH, 2009. 1239 S.

[Tec20] Techopedia Inc., Hrsg. *Build*. 2020. URL: https://www.techopedia.com/definition/3759/build (besucht am 12. 11. 2020).

[The19] The Apache Software Foundation, Hrsg. *PHP*. Apache httpd Dokumentation. 22. Mai 2019. URL: https://cwiki.apache.org/confluence/display/HTTPD/php (besucht am 12. 11. 2020).

[The20a] The Apache Software Foundation, Hrsg. *Apache HTTP Server Project*. 2020. URL: https://httpd.apache.org (besucht am 07. 10. 2020).

[The20b] The Apache Software Foundation, Hrsg. *The Apache Tomcat Connectors - AJP Protocol Reference*. Version 1.2.48. 9. März 2020. URL: https://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html (besucht am 01. 10. 2020).

[The20c] The Linux Foundation, Hrsg. *Assign Pods to Nodes using Node Affinity*. Version vom 30. Mai 2020. Kubernetes Dokumentation. 2020. URL: https://kubernetes.io/docs/tasks/configure-pod-container/assign-pods-nodes-using-node-affinity/ (besucht am 18. 11. 2020).

[The20d] The Linux Foundation, Hrsg. *Kubernetes Components*. Version vom 28. Aug. 2020. Kubernetes Dokumentation. 2020. URL: https://kubernetes.io/docs/concepts/overview/components/ (besucht am 19. 11. 2020).

[The20e] The PHP Group, Hrsg. *FastCGI Process Manager (FPM)*. PHP Dokumentation. 2020. URL: https://www.php.net/manual/en/install.fpm.php (besucht am 06. 10. 2020).

[Tia+19] Dave (Jing) Tian, Joseph I. Choi, Grant Hernandez, Patrick Traynor und Kevin R. B. Butler. "A Practical Intel SGX Setting for Linux Containers in the Cloud". In: *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*. CODASPY 2019. New York City, New York, USA: Association for Computing Machinery, März 2019, S. 255–266.

[Tia+18] Hongliang Tian, Qiong Zhang, Shoumeng Yan, Alex Rudnitsky, Liron Shacham, Ron Yariv und Noam Milshten. "Switchless Calls Made Practical in Intel SGX". In: *Proceedings of the 3rd Workshop on System Software for Trusted Execution*. SysTEX 2018. New York City, New York, USA: Association for Computing Machinery, 15. Okt. 2018, S. 22–27.

[Tia+17] Hongliang Tian, Yong Zhang, Chunxiao Xing und Shoumeng Yan. "SGX-Kernel: A Library Operating System Optimized for Intel SGX". In: *Proceedings of the Computing Frontiers Conference*. CF 2017. New York City, New York, USA: Association for Computing Machinery, Mai 2017, S. 35–44.

[TRA15] Andrea Tosatto, Pietro Ruiu und Antonio Attanasio. "Container-Based orchestration in cloud: state of the art and challenges". In: *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*. CISIS 2015. Blumenau, Santa Catarina, Brasilien: The Institute of Electrical und Electronics Engineers, Inc., Juli 2015, S. 70–75.

[Toz16] Christopher Tozzi. *Making the Migration to Containers Easier*. Hrsg. von MediaOps Inc. Container Journal. 19. Juli 2016. URL: `https://containerjournal.com/features/migrating-containers-hard-make-easier/` (besucht am 11. 11. 2020).

[TPV17] Chia-Che Tsai, Donald E. Porter und Mona Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX". In: *2017 USENIX Annual Technical Conference*. USENIX ATC 2017. Santa Clara, California, USA: USENIX Association, Juli 2017, S. 645–658.

[Vau+18] Sébastien Vaucher, Rafael Pires, Pascal Felber, Marcelo Pasin, Valerio Schiavoni und Christof Fetzer. "SGX-Aware Container Orchestration for Heterogeneous Clusters". In: *2018 IEEE 38th International Conference on Distributed Computing Systems*. ICDCS 2018. Wien, Österreich: The Institute of Electrical und Electronics Engineers, Inc., Juli 2018, S. 730–741.

[Wan+19] Huibo Wang, Erick Bauman, Vishal Karande, Zhiqiang Lin, Yueqiang Cheng und Yinqian Zhang. “Running Language Interpreters Inside SGX: A Lightweight, Legacy-Compatible Script Code Hardening Approach”. In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. Asia CCS 2019. New York City, New York, USA: Association for Computing Machinery, Juli 2019, S. 114–121.

[WBA17] Ofir Weisse, Valeria Bertacco und Todd Austin. “Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves”. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA 2017. New York City, New York, USA: Association for Computing Machinery, Juni 2017, S. 81–93.

[WK00] Steve Wilson und Jeff Kesselman. *Java Platform Performance. Strategies and Tactics*. Boston, Massachusetts, USA: Pearson Education, Inc., 2000. Kap. 3: Measurement Is Everything. 230 S.

[Win15] Christof Windeck. “Intel: Software Guard Extensions (SGX) erst mit den neuesten Skylake-Prozessoren”. In: *Heise Online* 10/2015 (7. Okt. 2015).

[Wir20] Wireshark Foundation, Hrsg. *Wireshark. Go Deep*. 2020. URL: `https://www.wireshark.org/` (besucht am 29. 10. 2020).

[Wor20a] WordPress Foundation, Hrsg. *Comments in WordPress*. WordPress Dokumentation. 2020. URL: `https://wordpress.org/support/article/comments-in-wordpress/` (besucht am 26. 10. 2020).

[Wor20b] WordPress Foundation, Hrsg. *Content Visibility*. WordPress Dokumentation. 2020. URL: `https://wordpress.org/support/article/content-visibility/` (besucht am 05. 10. 2020).

[Wor20c] WordPress Foundation, Hrsg. *How to install WordPress*. WordPress Dokumentation. 2020. URL: `https://wordpress.org/support/article/how-to-install-wordpress/` (besucht am 26. 10. 2020).

[Wor20d] WordPress Foundation, Hrsg. *REST API Handbook*. WordPress Dokumentation. 2020. URL: `https://developer.wordpress.org/rest-api/` (besucht am 05. 10. 2020).

[Wor20e] WordPress Foundation, Hrsg. *Server Environment*. Version vom 01. Nov. 2020. WordPress Dokumentation. 2020. URL: `https://make.wordpress.org/hosting/handbook/handbook/server-environment/` (besucht am 05. 11. 2020).

[Wor20f] WordPress Foundation, Hrsg. *WordPress Support*. 2020. URL: `https://wordpress.org/support/` (besucht am 05. 10. 2020).

[Wor17] WordPress REST API Team, Hrsg. *Basic Authentication handler*. Version vom 03. Dez. 2017. GitHub Repository. 2017. URL: `https://github.com/WP-API/Basic-Auth` (besucht am 26. 10. 2020).

[Xia+14] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato und Haiyong Xie. "A Survey on Software-Defined Networking". In: *IEEE Communications Surveys & Tutorials* 17.1 (13. Juni 2014), S. 27–51.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Elias Ullrich

Dresden, 26. November 2020