Tampere University

Muhammad Owais Javed

PESE
(PRIVACY EXTENSION FOR SEARCH ENGINES)

Development of PESE via Enarx framework for augmenting privacy and anonymity through TEE utilization

Master of Science Thesis
Faculty of Information Technology and Communication Sciences (ITC)
Examiners: D.Sc. Juha Nurmi
M.Sc. Arttu Paju
November 2023

ABSTRACT

Muhammad Owais Javed: PESE
(Privacy Extension for Search Engines)
Master of Science Thesis
Tampere University
Master's Programme in Information Technology
November 2023

Web search engines accumulate substantial quantities of personal data extracted from the queries of internet users. These datasets encompass a range of sensitive factors, including individual interests, engagements, health issues, work chronicles, religious principles, and political orientations. This aggregation of data components heightens the threat to user privacy, especially in situations characterised by data breaches, unauthorised access, or inappropriate exploitation of their private information.

The present thesis introduces a unique solution Privacy Extension for Search Engine (PESE), which combines the concepts of unlinkability and indistinguishability to enhance the privacy and anonymity of web users. Our solution employs a Trusted Execution Environment (TEE) to ensure the confidentiality of search queries, thereby thwarting any potential monitoring or interception by search engines and external entities. TEE consists of a secure processor component that offers selected programmes executing on the processor with data security, secrecy, as well as integrity. It effectively isolates these applications from the Rich Execution Environment (REE).

We utilize Enarx, an open-source deployment framework for designing and building our TEE application. It provides a robust and flexible environment that facilitates the secure execution of workloads within a TEE. Enarx's flexibility in accommodating different hardware architectures and platforms allows our PESE solution to be easily deployed across diverse systems without significant adjustments. This wide-ranging compatibility increases the applicability and accessibility of our solution, making it suitable for a wider array of environments.

Throughout this thesis, we expand on the implementation of PESE via TEE, elaborating on its operation's complexities. Furthermore, we conduct a thorough review of the Enarx deployment framework, highlighting its central role in the design and development of our TEE application.

Keywords: Enarx, TEE, Trusted Execution Environment, Trusted Computing, Confidential Computing, Confidentiality, Privacy, Anonymity, privacy-preserving

The originality of this thesis has been checked through the Turnitin Originality Check service. Furthermore, AI tools such as Grammarly, Quillbot, and ChatGPT have been used to improve the writing style of this thesis in accordance with Tampere University guidelines [150].

PREFACE

First and foremost, my heartfelt thankfulness to Almighty Allah for His blessings, compassion, guidance, and courage at every stage of my life. As I approach the completion of my Master's degree, I find myself to be blessed with numerous important people whose invaluable contributions in my life, have played key roles in influencing my journey.

I would like to express my special gratitude and admiration to my supervisors, Juha Nurmi and Arttu Paju, whose unwavering support, sage counsel, and enlightening guidance have been instrumental in moulding my personal and academic growth throughout this research endeavor.

I would also like to extend my appreciation to my program professor, Professor Billy Bob Brumley, for his encouragement during my academic journey. My sincere thanks also go to my esteemed colleagues at the Network and Information Security Group (NISEC) - Nicola, Ale, Antti, Marko, and Juha S., with whom I have had the honour to collaborate.

I am eternally grateful to my parents for their enduring love and support through both the highs and lows of life and their consistent prayers. Every success I attain, I dedicate to you, Mama and Baba for you have been the unwavering wind beneath my wings. I would like to thank my elder brother, Annus Javed, for always providing support during times of adversity.

I would especially want to thank Bakhtawar, who has always been there for me like a sister to support me. Usman, Shahram, and Saad, your insightful reminders have always served as a profound source of inspiration. Moreover, I will always value the influence of those who showed me care, love, and support while I was pursuing my master's.

Tampere, 18th November 2023

Muhammad Owais Javed

CONTENTS

1. Introduction 1
1.1 Motivation and Goals 2
1.2 Objectives and Contributions 2
1.3 Research Questions 3
1.4 Structure 5
2. Background 7
2.1 Trusted Execution Environment 7
2.2 TEE Applications 9
2.3 TEE Architecture 12
2.3.1 Secure Boot 12
2.3.2 Secure Scheduling 14
2.3.3 Inter-Environment Communication 14
2.3.4 Secure Storage 15
2.3.5 Trusted I/O Path 15
2.4 TEE Containers 16
2.4.1 Apache Teaclave 17
2.4.2 EGo SDK 18
2.4.3 Fortanix EDP 18
2.4.4 Mystikos 19
2.4.5 Gramine 20
2.4.6 Occlum 21
2.4.7 SCONE 22
2.4.8 vSGX. 23
2.4.9 Enarx. 23
2.5 Privacy and Anonymity 25
2.6 Brief History of Privacy and Anonymity 26
2.7 Evolution of Privacy and Anonymity System 29
2.7.1 Type 0: Remailing System 29
2.7.2 Type I: the Cypherpunk remailer 30
2.7.3 Type II and III: Mixmaster remailer 31
2.8 Modern Privacy and Anonymity System 32
2.8.1 Tor 32
2.8.2 I2P. 34
2.8.3 Tor vs I2P Comparison 35

2.8.4 Bitcoin Mixers . . . 36
3. The Enarx Framework . . . 40
3.1 Design Principles . . . 40
3.1.1 Trust Architecture . . . 42
3.1.2 Threat Model . . . 44
3.2 Requirements . . . 45
3.3 Core Components . . . 46
3.3.1 Attestation: . . . 47
3.3.2 Enarx API and Core: . . . 48
3.3.3 Enarx Runtime: . . . 48
3.3.4 Management: . . . 48
3.4 Working Method . . . 49
3.5 Vulnerabilities . . . 51
3.6 Practicalities . . . 51
4. Designing Privacy Extension for Search Engines . . . 53
4.1 Literature Review . . . 53
4.2 System and Adversary Model . . . 56
4.3 Objectives . . . 57
4.4 Design Approach and Methodology . . . 58
4.5 Limitations and Potential Bias . . . 60
5. Results of PESE Development and Testing . . . 62
5.1 Performance Evaluation . . . 62
5.1.1 Degree of Anonymity . . . 64
5.2 Practical Consideration and Potential Issues . . . 65
6. Conclusion . . . 67
6.1 Answers to Research Questions . . . 67
6.2 Future Research and Open Questions . . . 69
References . . . 71
Appendix A: Appendix . . . 86
A.1 PESE CODE 102

LIST OF FIGURES

1.1 Thesis Structure 4

2.1 REE and TEE Overview 8
2.2 TEE Applications Attribute Relationship 12
2.3 TEE Building Blocks 13
2.4 TCons Interface Design 16
2.5 Tor Network 33
2.6 Simplified I2P Network 34
2.7 High level Illustration of Bitcoin Mixture 37

3.1 Virtualization and Container Stack 41
3.2 Basic Layer Diagram of AMD SEV 43
3.3 Enarx Simplified Architectural Components and Integration 46
3.4 Enarx Architecture Components and Integration 47
3.5 Enarx Application Deployment Flowchart 49
3.6 Enarx Attestation Process Diagram 50

4.1 Design of Privacy Extension for Search Engines (PESE) 59

5.1 CPU Cycles Time Elapsed 63
5.2 User Time, System Time, RAM consumption plot 64

LIST OF TABLES

1.1 Mapping between Research Questions and corresponding Chapters 6

2.1 Trusted Containers Comparison 24
2.2 Comparison of Tor and I2P 36
2.3 Illustrations of Bitcoin Mixer functionalities in popular Bitcoin mixing services 38

3.1 Components and Enarx Trust Status 44

4.1 Relevant PESE developments 54
4.2 Private web search mechanisms comparison 56

5.1 PESE Environment Performance Metrics 63

GLOSSARY

API	Application Programming Interface
ASLR	Address Space Layout Randomization
CC licence	Creative Commons licence
CPU	Central Processing Unit
CSPs	Cloud Service Providers
DEP	Data Execution Prevention
DRM	Digital Right Management
EDP	Enclave Development Platform
EHRs	Electronic Health Records
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HIPAA	Health Insurance Portability and Accountability Act
HTTP	Hypertext Transfer Protocol
I2P	The Invisible Internet Project
ISA	Instruction Set Architecture
ISO	International Organization for Standardization
ISVs	Independent Software Vendors
MLaaS	Machine Learning as a Service
NISEC	Network and Information Security Group
OE	Open Enclave
OEMs	Original Equipment Manufacturer's
OS	Operating System
OTP	One Time Programmable
PGP	Pretty Good Privacy
RAM	Random Access Memory
REE	Rich Execution Environment
ROM	Read Only Memory

RPC	Remote Procedure Call
RPMB	Replay Protected Memory Block
SDK	Software Development Kit
SEP	Secure Enclave Processor
SEV	Secure Encrypted Virtualization
SGX	Software Guard Extension
SMTP	Simple Mail Transfer Protocol
TAs	Trusted Applications
TAU	Tampere University
TCB	Trusted Computing Base
TCons	Trusted Execution Environment Containers
TCP	Transmission Control Protocol
TDX	Trusted Domain Extensions
TEE	Trusted Execution Environment
Tor	The Onion Routing
TPM	Trusted Platform Module
TUNI	Tampere Universities
TZSW	Trust Zone Software
URL	Uniform Resource Locator
VM	Virtual Machine
VMM	Virtual Memory Manager
WASI	Web Assembly System Interface
Wasm	Web Assembly
WSEs	Web Search Engines

1. INTRODUCTION

The internet has seamlessly integrated into our daily lives, fundamentally transforming our modes of communication, information retrieval, and global interaction. The most prevalent action performed by Internet users is querying search engines, which enables them to efficiently access content from the tremendous quantity of data available on the Internet. When users conduct searches, they inadvertently reveal their interests and intentions, leaving behind a digital footprint. Search engines and potential eavesdroppers could use this information to generate user profiles and infer sensitive personal information. The user's expectations for their privacy and security are growing more rigorously; however as current systems continue to become more sophisticated and open, new difficulties continue to present themselves. Traditional security technologies are no longer efficacious [122]. Data confidentiality, privacy, and integrity solutions can not simply account for network-based attacks, but also those that emerge from a subgroup of software and hardware elements on the same platform or from an attacker with physical access [127].

A Trusted Execution Environment (TEE) represents a cutting-edge approach to confidential computing, offering reliable systems that surpass targeted defenses against specific attacks [121]. TEE is an isolated and restricted part of the main CPU whose primary purpose is to offer an additional layer of trust to programs executing within the processing unit. It operates independently and concurrently with the Rich Execution Environment (REE). The TEE possesses its isolated Read Only Memory (ROM), Random Access Memory (RAM), core processor(s), peripherals, cryptographic accelerators, and One Time Programmable (OTP) cryptographic assets, including hardware keys [13]. It meets every criterion for increased privacy, protection, and authenticity of data in the process. In TEE, when an application is authenticated, the untrusted modules load the trusted section into memory, and the trusted program is then protected from interference.

Several hardware manufacturers offer TEE implementations, such as Intel Software Guard Extensions (SGX), AMD Secure Encrypted Virtualization (SEV), ARM TrustZone, Intel Trusted Domain Extensions (TDX), and RISC-V Keystone [114]. Developing an application for TEE presents a formidable task, requiring the utilization of specific development frameworks or containers to streamline the process. One such platform-agnostic framework that facilitates the execution of sensitive workloads within TEE is Enarx [23]. It is an open-source project that aims to simplify the development and deployment of secure ap-

plications, enabling users to leverage the heightened security offered by TEE technology.

In this thesis, we aim to delve into various aspects of TEE and conventional privacy and anonymity solutions. In addition, we are going to delve into Enarx and its various aspects, including implementation and technical specifications. At last, we present Privacy Extension for Search Engines (PESE), specifically designed to operate within Enarx Keeper. The primary goal of PESE is to enhance user privacy and anonymity by concealing user online web queries. The forthcoming chapters of this thesis will offer a detailed exploration of TEE, Enarx, and PESE, encompassing their distinctive features, potential applications, limitations, and the potential ramifications on privacy and anonymity.

1.1 Motivation and Goals

Privacy is emerging as a significant concern for internet users, particularly in the context of their web search activities. Web search engines (WSEs) amass substantial quantities of user data, including search queries, browsing history, IP addresses, and device information [146]. This data can be used for the purpose of monitoring users' online activity, developing detailed profiles of their interests, and customizing advertising content accordingly.

One compelling solution to mitigate these privacy challenges involves the utilization of TEE. TEEs establish a secure and secluded enclave within the computing system, effectively segregating sensitive data and code from the operating system and other applications [44]. This isolation endows TEEs with remarkable resilience against unauthorized access, breaches, and data leaks. Consequently, TEEs hold significant potential in protecting user privacy and anonymity in web searches and online activities. Integrating TEE technology into WSE and associated digital infrastructure has the potential to create a more secure and privacy-conscious digital ecosystem.

The primary objective of this thesis is to create a solution that proficiently augments the privacy and anonymity of user's online search activities via integrating TEE. The overarching ambition is to thoroughly investigate TEE, Enarx, and subsequently implementation of PESE. Furthermore, a meticulous examination of the technical intricacies of PESE will be undertaken, including a comprehensive analysis of its distinctive characteristics and an assessment of its inherent limitations. Our overall objective is to empower individuals with enhanced capabilities to safeguard their personal information within the expansive online landscape.

1.2 Objectives and Contributions

This introduction serves as a comprehensive overview of the thesis, providing the reader with a concise map to visualize the structure and logical connections of the work. The

objective is to assist the reader in comprehending the significance and organization of the research and to provide a clear path for exploring the various topics presented in this thesis.

The map facilitates seamless navigation and augments the reader's comprehension of the research. Positioned beneath the designated topic are the pertinent research inquiries, which are subsequently followed by an exposition of the background information and a comprehensive overview of related research, thereby constituting the foundation upon which the research is conducted. Following the delineation of the background and related research, the subsequent section presents the research investigations and their corresponding outcomes, serving as the mechanism through which the research questions are addressed.

This thesis research is substantiated by our publication entitled *"A Systematic Review of TEE Usage for Developing Trusted Applications,"* which plays an important role in the pursuit of RQ1. The primary objective of our publication is to undertake a meticulous examination of the TEE ecosystem and its practical usability. To this end, the publication compiles a diverse array of academic and practical illustrations, synthesizing them to provide valuable insights into the evolution of TEE technology while concurrently conducting a thorough analysis of the tools accessible to developers. Additionally, the publication encompasses an evaluation of trusted container projects, performance testing, and the identification of prerequisites for application migration into these containers, all of which contribute substantively to the overarching thesis objective of selecting the Enarx framework for the development of PESE.

For the purpose of enhancing the reader's comprehension of the subject matter, this publication is thoughtfully included in the appendix. Furthermore, the inclusion of PESE code fragments is also included in the appendix demonstrates the commitment to open science principles and provides a concrete representation of the research conducted for the present thesis. The research questions that form the core of this thesis will be thoroughly explored in the subsequent section.

1.3 Research Questions

The research questions in this thesis are focused on the implementation and viability of PESE, a novel application based on TEE and constructed using the Enarx framework, with the objective of enhancing the privacy and confidentiality of web users. These questions can be articulated as follows:

RQ1. What is the importance of TEE in augmenting the security of data and operational processes?

RQ2. How do conventional solutions address privacy and anonymity concerns for

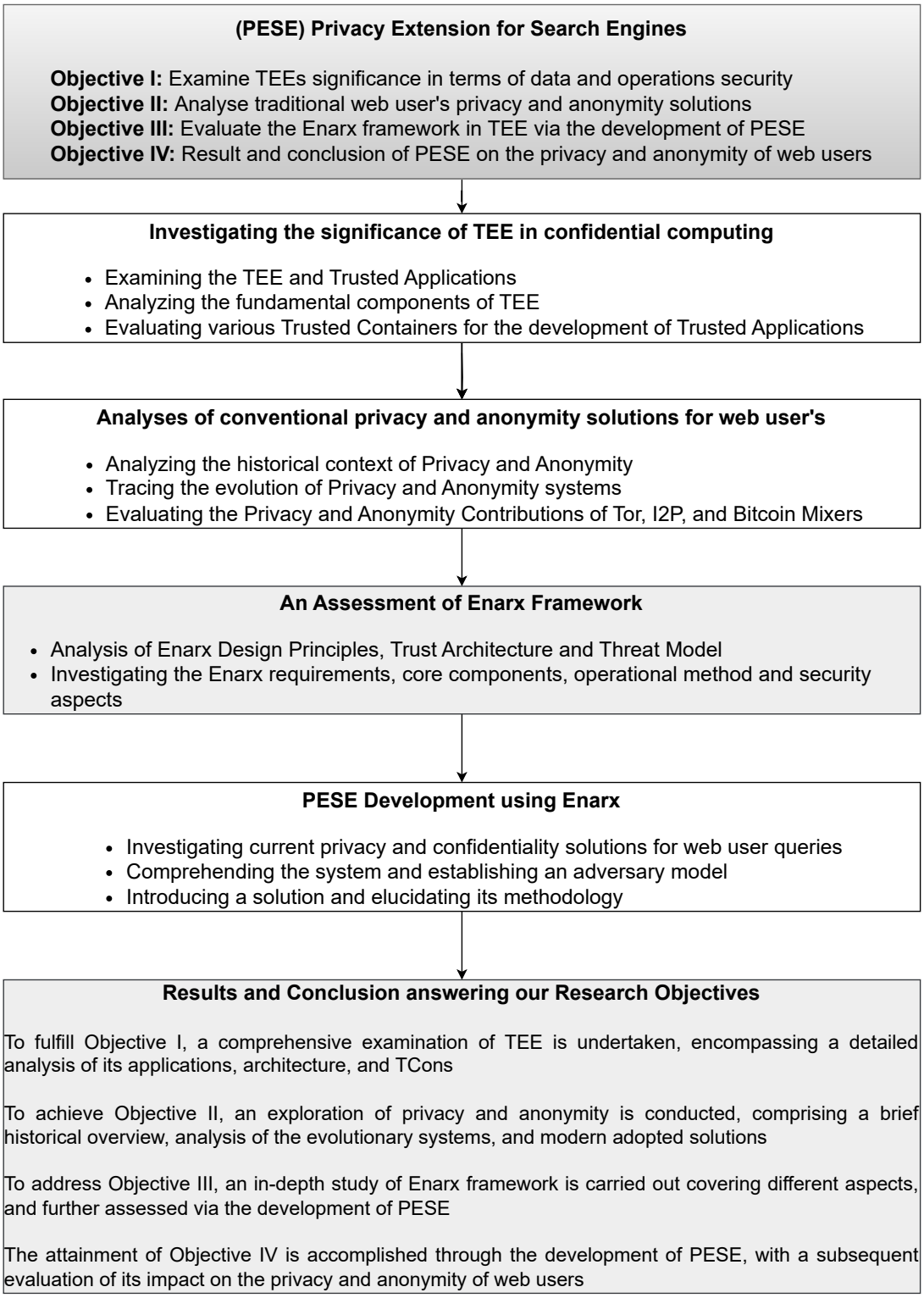

Figure 1.1. *A detailed map that visualize the structure and scope of the thesis*

web users?

RQ3. How effective is the Enarx framework for developing privacy-enhanced application PESE?

RQ4. What are the results and conclusions regarding the impact of PESE on the privacy and anonymity of web users?

The research questions of this thesis will be addressed as follows:

For RQ1: In Chapter 2, we will undertake an exhaustive examination of TEE, delving into its multifaceted importance. Our analysis will encompass an evaluation of TEEs role in enhancing the security and privacy of sensitive data and processes, as well as a comprehensive exploration of how TEEs function as a means to mitigate potential threats.

For RQ2: In addition to the detailed exploration of TEE, Chapter 2 provides a comprehensive overview of the conventional solutions that pertain to user privacy and anonymity. Within this chapter, we not only present these existing traditional solutions but also highlight the inherent challenges and complexities associated with their implementation.

For RQ3: Chapter 3 is dedicated to a comprehensive exploration of the core principles underlying the Enarx framework. Within this chapter, we will provide a detailed exposition of the framework's architectural intricacies and functional attributes, emphasizing its pivotal role in enabling the creation of TAs that harness the capabilities of TEE. Subsequently, in Chapter 4, we will delve further into the evolution of TAs, focusing on the development of PESE via the Enarx framework.

For RQ4: While Chapter 4 predominantly centers on the in-depth exploration of the development of PESE, Chapter 5 will be dedicated to the comprehensive assessment of PESE's effectiveness in augmenting the privacy and confidentiality of end users, Subsequently Chapter 6 will encapsulate the findings and conclusions derived from the evaluation of PESE, in conjunction with addressing other research objectives.

1.4 Structure

The present thesis is comprised of *Six* chapters, with the Chapter 1 serving as an introduction to the scope of the thesis. To enhance reader comprehension, Table 1.1 offers a structured representation illustrating the alignment between the research questions and the corresponding chapters where they are addressed.

In essence, this thesis offers a comprehensive exploration encompassing TEE, the Enarx framework, and the privacy and anonymity solution, PESE, designed to protect web users search activity. The research encompasses technical intricacies, real-world implementations, and Enarx's contributions to the development of PESE through the utilization of TEE technology. Furthermore, it presents a comprehensive evaluation of the project's findings

Research Questions	Answered in
RQ1 What is the importance of TEE in augmenting the security of data and operational processes?	Chapter 2, Section 2.1 - Section 2.4
RQ2 How do conventional solutions address privacy and anonymity concerns for web users?	Chapter 2, Section 2.5 - Section 2.8
RQ3 How effective is the Enarx framework for developing privacy-enhanced application PESE?	Chapter 3, Chapter 4
RQ4 What are the results and conclusions regarding the impact of PESE on the privacy and anonymity of web users?	Chapter 4, Chapter 5

Table 1.1. *Mapping between Research Questions and corresponding Chapters*

and acknowledges its limitations.

2. BACKGROUND

In this chapter, we present a synopsis of the current state of knowledge concerning TEE and Privacy and Anonymity which is significant to the objective of the thesis.

2.1 Trusted Execution Environment

The convergence of cutting-edge connectivity solutions, advanced sensing technology, and purposeful digital communities has sparked an unprecedented surge in the generation and utilization of personal data. Traditional manual processes, once limited in scope, such as banking, healthcare sectors and resource monitoring, have evolved into potent sources of interconnected digital information tied to individuals [10]. There has been a notable trend in sending data to cloud servers, leading to the dominance of cloud computing in the technology landscape [90]. Organizations now place a strong emphasis on robust security measures to safeguard sensitive code and data. Different sectors face unique challenges when it comes to ensuring data confidentiality, integrity, and privacy, ranging from financial transactions to citizen data and classified documents. Compliance with regulations such as HIPAA [68], GDPR [117], protection of intellectual property, and national security are crucial aspects of their data protection strategies. To maintain trust in customer and stakeholder relationships, organizations must prioritize security measures and innovative solutions to thrive in this environment.

To address the critical need for enhanced security, the integration of trusted computing concepts has become a prevailing trend, ensuring the protection of sensitive data in these dynamic architectures. Data can undergo protection in three distinct states: while at rest, during transit, and while in use [80]. While encrypting data at rest and during transit has become a standard procedure in cloud computing, the encryption of data in use, which constitutes the fundamental concept of Confidential Computing, remains an emerging concern [1].

Traditional trusted computing uses a dedicated hardware module called the Trusted Platform Module (TPM) [15] for platform security. However, it lacks the ability to execute third-party code in an isolated environment. A new approach allows tamper-resistant execution of arbitrary code within a confined environment. This environment is known by various names such as closed-box VM [54], TrustZone software (TZSW) [155] and trusted

language run time [126] however the term *Trusted Execution Environment (TEE)* coined by GlobalPlatform [58] is the one used in our thesis.

TEE is a cutting-edge security solution designed to create a secure zone within the central processing unit (CPU). Its primary function is to process data in a protected environment, completely isolated from REE operations [44]. REE also known as the *Normal World*, is the standard computing environment where most applications are developed for and deployed to [39], This isolation includes the hypervisor and the operating system, ensuring a high level of security. It possesses distinct processing, memory, and storage capabilities [17]. In comparison to the REE, TAs execute within a more secure environment due to the isolated nature of TEE [70]. This isolation is instrumental in preserving data privacy and integrity, irrespective of the individual with authorization to the host system in which an application is in operation.

The Figure 2.1 illustrates a device portrayed as a sequence of separate environments, each characterized by its distinctive attributes and functions. The terminologies introduced by GlobalPlatform [58] are used to describe the concepts illustrated in the Figure 2.1.

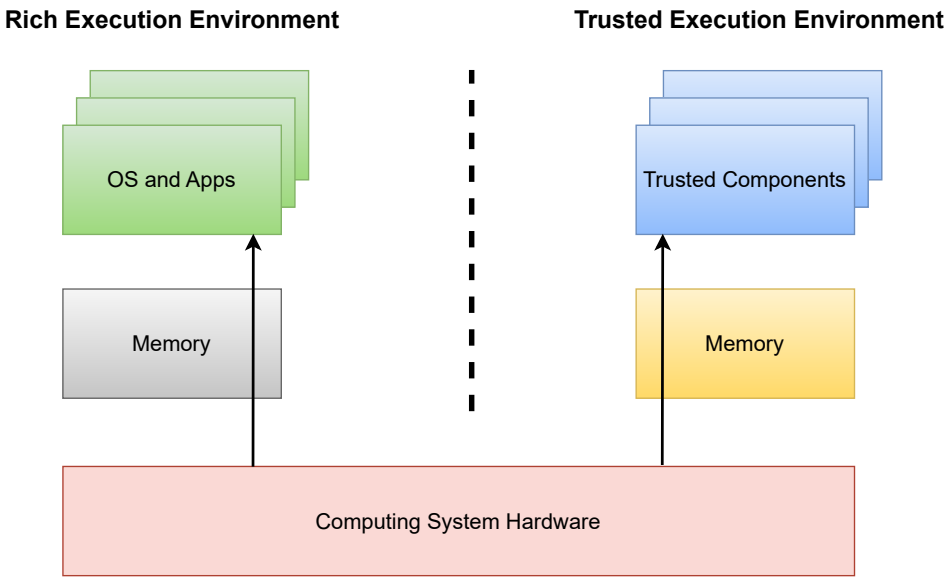

Figure 2.1. *Generic overview of REE and TEE*

One of the key benefits for developers is the ability to design REE applications and services that maintain an adequate level of security, even when the OS is compromised. This resilience is achieved as sensitive operations are confined within the TEE, effectively protecting confidential data such as cryptographic keys. The TEE utilizes security mechanisms [25] rooted in hardware, including encryption and secure boot, to establish an advanced and secure context for the execution of applications, execution of transac-

tions, and storage of confidential information. By doing so, TEE offers a reliable and effective solution to enhance overall data protection.

2.2 TEE Applications

In complex scenarios, involving multiple entities with varying levels of trust, the implementation of hardware-backed TEEs provides a strong and specialized environment to safeguard sensitive operations and data, even when interacting with potentially untrusted parties. As a result, TEE provides a valuable foundation for building secure and privacy-preserving applications in complex, multilateral settings [139]. TEE applications exhibit diverse classifications, but the most prevalent usage scenarios are as follows:

Cloud Computing: Cloud computing has revolutionized how users store and access their data, but it entails certain trade-offs, including the loss of direct control over the physical infrastructure where data is stored [67]. This lack of direct management raises privacy concerns, especially when dealing with sensitive data, which poses a significant drawback of cloud systems [56]. In order to effectively tackle these difficulties and provide a secure cloud computing environment, TEEs have garnered increasing attention due to their ability to offer a specialised security solution that safeguards against possible dangers arising from compromised infrastructure. By restricting the adversary's access to specific sections of the TA, TEEs protect sensitive code and data from unauthorized access and tampering. The improved security measures provide significant assurances about privacy, hence promoting wider acceptance of cloud services. Users may be certain that their data will be protected and kept secret inside the cloud environment.

Financial payments: Digital wallets, online payment gateways, cryptocurrency wallets, and the utilization of computing devices as point-of-sale terminals all have well-stated security requirements. Blockchain technology also finds application in both online payments and enhancing transaction security, as it enables secure and transparent recording of transactions across multiple nodes in a network. TEE integrating with Blockchain can play a central role in protecting sensitive information, encompassing user credentials, payment card details, and transaction data [93]. This is achieved through a secure hardware channel between the TEE and the payment application, effectively encrypting and isolating data from the device's regular system and other applications. Functioning as a trusted backend systems, TEE elevates the security of financial transactions, necessitating users to input a PIN, password, or biometric identifier. This comprehensive approach substantially reduces the risk of data breaches and unauthorized access, thereby fortifying overall transaction security.

Content protection: TEE provides a secure environment for Digital Rights Management (DRM) systems, and copyright holders use TEE to protect digital content, such as movies, music, and e-books, from unauthorized access and piracy [105]. Furthermore, the estab-

lishment of a secure hardware link between the TEE and the device's display effectively restricts the device owner's ability to get access to any confidential information held inside the TEE. TEEs play a vital role in ensuring that digital content is only accessible by authorized users and connected devices [16, 44].

Healthcare Data: Effectively managing medical information, encompassing electronic health records (EHRs), telemedicine data, and sensitive outputs from body sensors, poses substantial challenges when processing on third-party untrusted clouds while preserving user privacy. Furthermore, the continuous and voluminous nature of data generated by medical sensors necessitates efficient and secure processing, even under strong adversarial models. Fortunately, the emergence of consumer-grade processors with TEE offers a promising solution, surpassing less flexible approaches like homomorphic encryption [130]. TEE plays a pivotal role in ensuring patient privacy protection and maintaining the confidentiality of medical information.

Authentication: TEE has emerged as a crucial component for enhancing authentication processes, especially with regard to biometric identity methods. Biometric data, such as facial recognition and fingerprints, are processed and stored securely within the TEE, effectively preventing unauthorised access. In addition, the TEE supports cryptographic private keys, thereby facilitating the implementation of authentication standards that do not require a password. TEEs transfer the authentication locus from remote servers to user devices by creating a secure and isolated environment, resulting in authentication experiences that are not only convenient, reliable, and theft-resistant [19]. This paradigm shift accomplishes robust security properties comparable to traditional two-factor authentication without requiring a separate device [118].

Run-time Verification: In the context of cryptographic protocol implementation, vulnerabilities and malware attacks that target specific runtime can compromise the security of an otherwise proven safe protocol design. Despite proven safe protocol designs, incorrect implementation and runtime attacks can compromise security. TEEs enable runtime-verification approaches to continuously monitor protocol execution, detecting anomalies in real-time [36, 85]. Additionally, TEEs create secure environments for critical components like kernel binaries, ensuring they remain isolated and protected from potential attacks. In the occurrence of an unauthorised breach, the security monitoring service within the TEE can promptly shut down, preventing further exploitation and safeguarding the application's integrity [18]. Leveraging TEE for run-time integrity enhances the security of cryptographic protocols, providing a robust defence against real-time threats and reinforcing the trustworthiness of applications even under challenging circumstances.

Machine Learning: The increasing demand for Machine Learning as a Service (MLaaS) provided by prominent cloud providers such as Amazon AWS, Microsoft Azure, and Google Cloud [22] has raised concerns about protecting user input privacy. Deploying

pre-trained models in the cloud requires strong measures to maintain the confidentiality of user data, particularly during inference. While conventional methods like data encryption address data protection during storage and communication, TEE provide a promising approach to ensuring data security throughout computational processes. By leveraging a combination of hardware and software techniques, TEE effectively segregate and protect confidential data and computational processes. This approach presents a resilient and all-encompassing strategy to augment privacy and security in the domain of machine learning inference inside cloud-based settings [96].

Identity Management: The prevalence of online platforms that collect, share, and monetize user data has raised concerns about the privacy and fairness of users. By analyzing this data, these platforms infer user behaviours and preferences, generating detailed user profiles [107]. However, TEE offers a viable solution to safeguard user confidentiality and ensure fairness among different entities involved [115]. Various applications, including those managing digital identities, national ID systems, and digital signature systems, can benefit from TEE secure environment to protect digital identities and thwart unauthorized access, thereby enhancing overall security and privacy measures.

Secure Modular Programming: In the context of secure modular programming, TEEs allow developers to partition sensitive components and critical operations into separate trusted modules. These modules can be executed in a secure and isolated environment, away from the potentially less secure REE. As a result, TEEs provide a strong foundation for building applications with enhanced security, confidentiality, and integrity [57]. Using TEEs in secure modular programming ensures that critical functions, such as cryptographic operations and key management, remain protected from potential attacks and unauthorized access. It also helps prevent information leakage and tampering with sensitive data during program execution. Furthermore, TEEs offer a standardized interface for interacting with the secure modules, allowing developers to integrate their secure functionalities seamlessly into the overall application. This promotes code reusability, ease of maintenance, and efficient development of secure applications.

TEE offers a highly secure environment for diverse applications, leveraging hardware-based security features to create an isolated space separate from the primary operating system and other software on the device. Our publication on "A Systematic Review of TEE Usage for Developing Trusted Applications" presents Figure 2.2, which illustrates various usage examples, highlighting the significant relationships between categories, mechanisms, and security properties. Through these relationships, TEE effectively protect sensitive data and computations, fortifying the security stance of applications that employ them.

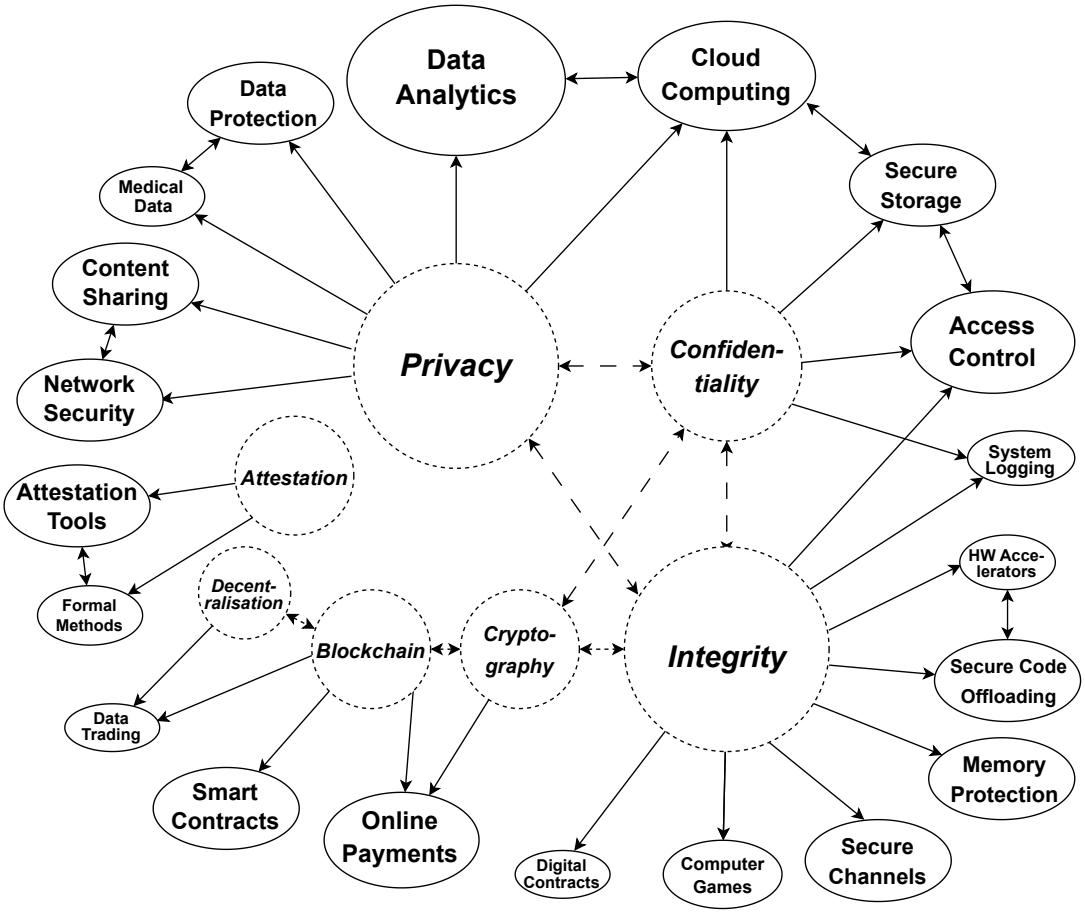

Figure 2.2. *Relationships between TEE usage, categories, security properties, and mechanisms*

2.3 TEE Architecture

TEE implementations utilize a range of technologies to establish hardware-based isolation and ensure secure execution environments. Mohamed Sabt, Mohammed Achemlal and Abdelmadjid Bouabdallah [122] have identified and described the essential building blocks of TEE which are presented in the Figure 2.3. This section offers a concise summary of the functioning and operation of each of the aforementioned blocks.

2.3.1 Secure Boot

A secure bootstrap process is essential for the trusted OS kernel [12]. By breaking down the bootstrap process into discrete steps, it becomes evident that if an adversary gains control over any step, subsequent steps cannot be trusted [69].

Secure Boot plays its role during the boot-time process that ensures only trusted software components are loaded and executed on a device. The process begins when the device is powered on and involves verifying the authenticity of each component in the

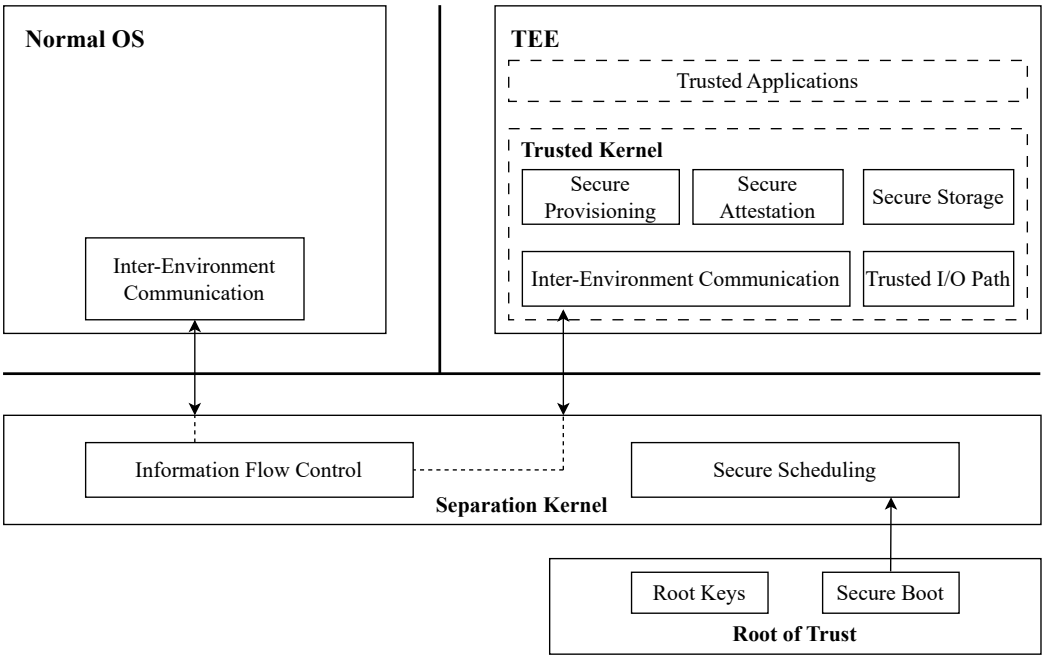

Figure 2.3. *Fundamental Components of TEE Architecture*

boot chain, starting with the boot loader. It then proceeds to validate the authenticity of the operating system and any other loaded software components. If any unauthorized or altered component is detected in the boot chain, the Secure Boot process halts the booting procedure, preventing the execution of malicious software.

By preventing the execution of unauthorized software, such as malware, viruses, and other malicious code, Secure Boot safeguards the security and integrity of the TEE environment. This protective measure ensures the confidentiality and protection of sensitive data and applications against potential malicious attacks and unauthorized access [122].

The effectiveness of Secure Boot is mostly contingent upon the root of trust, which often encompasses a secure element that is seamlessly integrated inside the hardware of the TEE. The root of trust plays a crucial role by verifying the authenticity of each component in the boot chain, thereby allowing only trusted components to be loaded and executed [30].

As a key component of TEE security, Secure Boot plays a pivotal role in ensuring that only trusted software components are executed within the TEE environment. It verifies the authenticity of each component in the boot chain, effectively preventing the execution of unauthorized software.

2.3.2 Secure Scheduling

Secure Scheduling is integral to the separation kernel, ensuring isolated and secure task execution in a TEE. It plays a crucial role in coordinating the TEE and the system, maintaining the responsiveness of the main OS while protecting the TEE tasks by preemptively designing the scheduler and considering real-time constraints [124]. Secure Scheduling guarantees the secure execution of critical tasks, such as cryptographic operations and sensitive data processing. It prevents interference from other tasks or the underlying OS, ensuring accurate and trustworthy task results even in the presence of malicious or faulty software.

To implement Secure Scheduling, dedicated hardware resources and memory partitions are allocated to the TEE. This hardware-based isolation protects the TEE's resources from interference and contributes to its robust security. Secure Scheduling's crucial role in resource allocation and task scheduling ensures that the TEE meets its security requirements, even in the presence of potentially malicious or unreliable software components. This mechanism is indispensable in maintaining the integrity and confidentiality of sensitive operations within the TEE [122].

2.3.3 Inter-Environment Communication

Inter-environment communication is one of the fundamental blocks of the security of TEE as it facilitates secure and regulated data interchange among various TEE environments. This guarantees the protection of sensitive data and adherence to TEE security prerequisites.

Inter-environment communication employs secure channels such as secure pipes or messaging to enable secure interaction with the rest of the system. These channels ensure data remains protected against interception and tampering during transmission and restrict access to authorized entities exclusively.

All Inter-environment communication methods must fulfil three vital characteristics: dependability (ensuring memory/time isolation), minimal overhead (avoiding unnecessary data duplication and context changes), and preservation of communication structures [122].

Despite offering advantages, Inter-environment communication introduces new threats, including message overload attacks [116], user and control data corruption attacks [29], memory faults caused by shared page removal, and unbound waits due to noncooperation of the untrusted part of the system.

In the paper [122], three communication models have been identified: GlobalPlatform TEE Client API [58], secure RPC (Remote Procedure Call) of Trusted Language Runtime

[126], and real-time RPC of SafeG [125]. Additionally, secure inter-environment communication is proposed in [74].

Inter-environment communication establishes a secure interface for TEE to interact with the system, enabling collaboration and data exchange between different TEE environments while ensuring the security and protection of sensitive data. Nevertheless, careful implementation and adherence to key attributes like reliability, minimal overhead, and preservation of communication structures are imperative to mitigate potential threats.

2.3.4 Secure Storage

Secure Storage guarantees the confidentiality, integrity, and freshness of stored data while restricting access to authorized entities only [88]. It is typically implemented in hardware with dedicated resources and memory partitions, isolating sensitive data from the underlying OS and other software components.

Sealed storage is a common approach for implementing secure storage [104], relying on three components: an integrity-protected secret key accessible only by the TEE, cryptographic mechanisms like authenticated encryption algorithms, and a data rollback protection mechanism like replay-protected memory blocks (RPMB) [145].

The utilization of secure storage ensures that sensitive data remains secure and isolated, shielding it from potential risks like malware or data breaches. The secure environment ensures the data confidentiality and security of sensitive data and applications, even in the presence of malicious or faulty software. Moreover, Secure Storage serves beyond data protection, offering a secure repository for configuration data such as security policies and access control rules, contributing to the controlled operation of the TEE.

Secure Storage plays a crucial role in TEE security, providing an encrypted and segregated area for data storage and ensuring the integrity and confidentiality of applications. Its implementation through sealed storage with integrity-protected secret keys, cryptographic methods, and data rollback protection enhances the overall security of the system [122].

2.3.5 Trusted I/O Path

The Trusted I/O Path protects the authenticity of communication between TEE and peripherals, ensuring that both input and output data remain safeguarded against eavesdropping and manipulation by malicious applications [2]. It also offers an optional level of confidentiality for such communication, adding further protection for sensitive data like keyboard inputs and sensor data. The Trusted I/O Path serves as a defence against four categories of attacks: screen capture, key logging, overlaying, and phishing attacks.

By protecting against these threats, it ensures the integrity of data being communicated between TEE and peripherals.

Furthermore, the Trusted I/O Path expands TEEs capabilities by providing a trusted path to user-interface devices. This feature enables direct interaction between human users and applications running inside the TEE, allowing for broader functionality and improved user experience within the secure environment [83, 84, 55]. Nevertheless, Trusted I/O Path is a crucial component of TEE security, ensuring the authenticity and confidentiality of communication with peripherals. Its protection against various attack vectors strengthens the overall security of the system, while its integration with user-interface devices enhances the user's ability to interact directly with secure applications [122].

2.4 TEE Containers

To ensure the compatibility of an application with various TEE technologies, it is important to conform to design solutions that are particular to the framework. However, the widespread adoption of TEE technologies can face challenges due to their limited support for unmodified programs. Specifically, users often encounter the need to integrate the corresponding hardware SDK (Software Development Kit) into their original programs to execute them inside a TEE instance, a process that demands considerable effort. Moreover, developers must implement attestation to establish trust in the application.

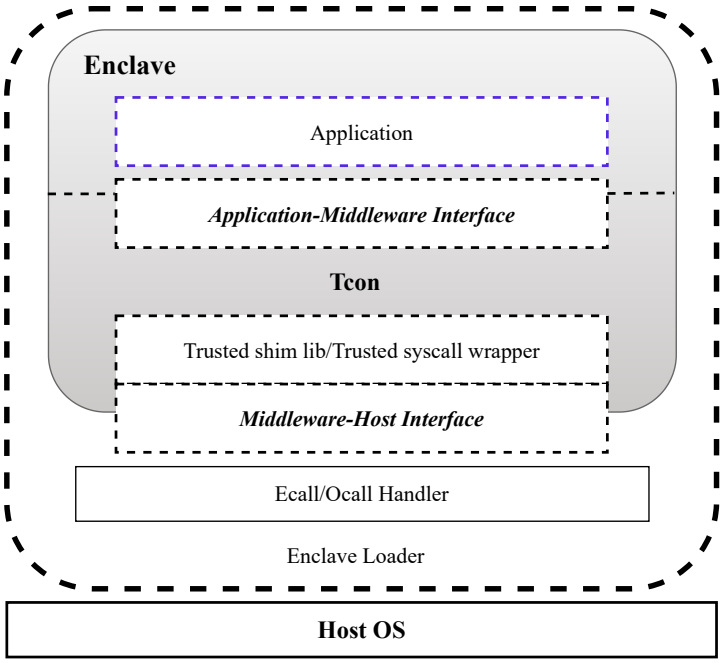

Figure 2.4. *Common TCons Interface Design*

To address this usability challenge, a solution has emerged in the form of container-style TEE middleware called TCons. These middleware options offer two approaches:

one involves the direct running of unaltered binary code inside a TEE or automatically modifying source code prior to its incorporation into a TEE executable. This innovation aims to streamline the process and enhance the usability of TEE technologies. Figure Figure 2.4 depicts the most typical TCons interface designed based on the paper [86].

It's worth noting that while there are various TCons available, many of them are no longer active. In the following, we will briefly go through the TCons that remain active in the year 2022 according to our publication. This overview will shed light on the available options for developers seeking to implement trustworthy and efficient applications within TEE environments.

2.4.1 Apache Teaclave

Apache Software Foundation introduced Apache Teaclave, an open-source project that focuses on secure and privacy-preserving computing using TEE. The primary aspect of Teaclave's architectural design is the use of TEE technologies, namely Intel SGX and ARM TrustZone. These technologies provide isolated enclaves that possess the capability to safely execute code [136]. This prevents unauthorized access to the data and code being processed, ensuring the utmost confidentiality and integrity of sensitive information throughout the computation process.

The primary objective of Apache Teaclave is to enable developers to perform data processing tasks in a secure environment. This security of data ensures that sensitive information remains protected at all times. Furthermore, Teaclave supports programming languages, including C/C++ and Python to cater to a broader community.

Teaclave also provides a secure method for executing and scheduling duties within the enclave, ensuring that data is properly isolated and protected [11]. This feature significantly reduces the possibility of data loss or unauthorised access, enhancing the overall security of the project. Moreover, Teaclave incorporates attestation mechanisms that verify the integrity of the enclaves. This process helps establish trust between different parties, ensuring that the enclaves are genuine and have not been tampered with. The maintenance of the security and dependability of the whole system is contingent upon the significance of this particular feature.

While Teaclave's native support for TEEs offers a secure execution environment, it also introduces challenges concerning application deployment and portability. To address these challenges, TEE Containers encapsulate Teaclave applications along with their runtime environment, libraries, and dependencies. This approach ensures seamless deployment and execution across different platforms, making the project more versatile and user-friendly [142].

By packaging the entire Teaclave application and its dependencies, developers can de-

ploy their applications on various systems without worrying about compatibility issues or potential security breaches. This advancement enhances the usability and efficiency of secure computing within Teaclave's TEE-enabled environment, opening up new possibilities for secure and privacy-preserving data processing.

2.4.2 EGo SDK

The EGo SDK is an open-source framework designed for the construction of secure applications using the Go programming language. These apps run in secure execution environments called enclaves, which are TEE instances strongly isolated, runtime encrypted, and attestable on Intel processors with SGX feature [43]. EGo builds on top of the Open Enclave (OE) SDK, providing an in-enclave Go library for attestation and sealing [91]. The compiler ensures your code runs inside an enclave, and the library offers access to key enclave features like Remote attestation and Sealing.

The entire application runs inside the enclave, with hidden transitions between trusted and untrusted code through the EGo runtime. The programming model eliminates the need to learn new concepts, simplifying confidential app development without partitioning efforts [138]. Many existing Go applications run on EGo without modification, making it easier to create server applications dealing with sensitive data like cryptographic keys or payment data, similar to HashiCorp Vault. However, there are limitations such as single-process restriction, experimental Cgo support [1], and unsupported stripped executables.

2.4.3 Fortanix EDP

Fortanix Enclave Development Platform (EDP) is a developer platform designed to simplify the development and deployment of applications using TEEs [51]. It is built around Intel SGX, a specific implementation of TEE technology available on certain Intel processors [72]. Leveraging SGX, Fortanix EDP enables applications to create isolated enclaves, securing sensitive operations from exposure to the outside world and the underlying operating system. The platform is designed for SGX, security, services, portability, and ease of use, and it has been battle-tested.

Fortanix EDP provides a comprehensive SDK that allows developers to build and integrate secure enclaves into their applications. It is completely open source and fully integrated with the Rust language compiler, allowing native Rust code to be compiled for enclaves without modifications [50]. The combination of Rust and Intel SGX enhances application security, safeguarding against development vulnerabilities and outsider attacks.

Fortanix EDP has gained rapid adoption among researchers and Independent software

[1] https://pkg.go.dev/

vendors(ISVs) in the runtime encryption community, empowering the creation of new products and services for confidential computing. Fortanix EDP offers a range of key features that facilitate secure enclave development and execution. The SDK and runtime provide a comprehensive set of tools and libraries to create and run enclaves securely. The platform ensures the integrity of enclaves through remote attestation, which verifies their authenticity to remote parties, establishing trust in the code running within the enclave [21].

Addressing the critical need for safeguarding sensitive data, Fortanix EDP includes secret management facilities. These tools allow for the secure generation, storage, and management of cryptographic keys and other confidential information within enclaves. One of the core functionalities of Fortanix EDP is protected execution, enabling vital sections of applications to be executed within secure enclaves. This protective measure ensures that sensitive data and code are shielded from potential threats and unauthorized access. Moreover, the platform caters to the need for secure communication and data exchange between multiple enclaves with its Inter-enclave communication support. This feature ensures that sensitive information can be shared among enclaves securely, reinforcing the overall security posture of applications leveraging Fortanix EDP [49].

2.4.4 Mystikos

Mystikos, an advanced runtime and suite of tools, has been purposefully developed to facilitate the seamless execution of Linux applications within TEE. Currently, the primary focus of the existing release centres around providing support for Intel SGX, while future versions are envisaged to extend this support to other TEEs.

The overarching objectives of Mystikos focus on efficient key management, attestation, hardware roots of trust, and data encryption for data at rest and in transit. By providing robust protection against various threats, Mystikos aims to streamline the migration of native or containerized applications into TEEs with minimal adjustments, making it easier to implement secure environments for a wide range of applications.

At the heart of Mystikos lies the empowerment of users and application developers, offering them authority over the Trusted Computing Base (TCB) composition. To achieve this, the project ensures that all execution environment components within the TEE are open-sourced with permissive licenses, promoting transparency and enabling easy customization for enhanced control [102].

Additionally, the project strives to expedite the re-targeting of other TEE architectures through an adaptable plugin architecture, simplifying the adaptation and deployment of applications to varying hardware environments. Mystikos has a sophisticated architec-

ture, with essential components such as a C-runtime based on musl libc [2] and designed to be compatible with glibc [3]. A lib-os like kernel is also integrated, alongside the kernel-target interface (TCALL) and a command-line interface, complemented by various related utilities.

Presently, Mystikos offers two target implementations: the SGX target, which relies on the Open Enclave SDK, and the Linux target, intended for verification on non-SGX platforms. Within the TEE, the minimalist kernel of Mystikos expertly manages essential computing resources, encompassing CPU/threads, memory, files, networks, and more. While it adeptly handles most syscalls akin to a regular operating system, certain limitations exist. Some syscalls are directly managed by the kernel, while others are appropriately delegated to the specified target during the launch of Mystikos, ensuring precise execution and compatibility [5].

2.4.5 Gramine

Gramine, formerly known as *Graphene*, serves as a library OS framework designed to ensure the secure execution of diverse Linux applications within an enclave powered by SGX [144]. Leveraging the capabilities of SGX technology, Gramine facilitates the operation of applications within an isolated environment, offering advantages akin to running a complete OS in a virtual machine. These benefits encompass guest customization, seamless portability to different host OS, and process migration [63].

A distinguishing feature of Gramine lies in its ability to enable applications to operate without necessitating any modification or rebuilding. To leverage the synergy of Gramine and SGX, developers are required to create a manifest file detailing the application's configuration, authorized interactions with the untrusted environment, permitted file access, and other pertinent information. When an application runs, Gramine verifies the authenticity and integrity of the manifest file and then uses it to load the application and its dependencies. Moreover, at runtime, Gramine efficiently resolves application requests to the host OS (system calls).

As of now, Gramine exclusively supports CPU-based interactions with the untrusted environment. Consequently, applications within a Gramine SGX enclave can proficiently execute CPU and memory-intensive tasks, read and write files, send and receive network packets, and communicate with other SGX enclaves. However, the framework does not currently support offloading computations to the GPU [81].

[2] https://musl.libc.org/
[3] https://www.gnu.org/software/libc/

2.4.6 Occlum

Occlum represents an innovative and cutting-edge open-source TEE that places a strong emphasis on ensuring high-assurance data and compute security. Leveraging the inherent hardware-level security features of modern CPUs, Occlum adopts a diverse range of software security approaches to guarantee the secure execution of applications. One of the primary advantages of Occlum lies in its ability to establish a safe computing environment entirely segregated from the underlying operating system and hardware. This means that even in the event of an attacker compromising the OS or hardware, access to sensitive data or execution of code within the TEE is thwarted. This feat is accomplished through the utilization of hardware-level security technologies like Intel SGX, which facilitates the creation of isolated and secure memory enclaves [134].

Occlum's ability to seamlessly adapt to a wide variety of applications is its most notable quality, making it a crucial and versatile solution. The TEE has been thoughtfully designed to be compatible with multiple programming languages, including C, C++, and Python, enabling both user-space and kernel-space application execution. This remarkable versatility empowers the development of various secure applications, encompassing secure email clients, web browsers, encrypted storage solutions, and privacy-preserving data analytics tools. To provide robust protection against internal and external threats, Occlum relies on a range of software-based security measures. These include the implementation of Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP) to counteract attackers attempting to exploit different vulnerabilities [158]. Additionally, the device incorporates secure boot methods, guaranteeing the execution of only trusted software.

Occlum encompasses key features that make it a powerful solution for running security-critical applications within SGX enclaves. One of its most notable features is Efficient Multitasking, achieved through lightweight LibOS processes that share the enclave's single address space. Additionally, Occlum offers support for Multiple File Systems, catering to different security requirements. These include read-only hashed FS, providing integrity protection, writable encrypted FS for confidentiality protection, and untrusted host FS, facilitating seamless data exchange between the LibOS and the host OS. Memory Safety is a top priority for Occlum, which has been developed from scratch using the Rust programming language. By doing so, it effectively minimizes low-level memory-safety bugs, ensuring heightened reliability and trustworthiness when hosting critical applications.

Furthermore, Occlum prioritizes user-friendliness with its Ease of Use features. The platform offers intuitive build and utility command-line tools, simplifying the process of executing applications on Occlum within SGX enclaves. By making the execution process straightforward and accessible through several shell commands, Occlum empowers developers to harness the benefits of secure enclaves without unnecessary complexity [99].

Occlum stands as a robust and adaptable tool tailored to the requirements of both enterprises and individuals seeking heightened levels of data and processing security. Whether the objective is to design a secure email client, an encrypted storage solution, or a privacy-preserving data analytics tool, Occlum provides the means to create the most secure and reliable applications possible.

2.4.7 SCONE

SCONE is an advanced platform that leverages Intel SGX technology to create and execute secure applications. The major objective of the system is to guarantee the continuous encryption of all data, including data at rest, data in transit, data in primary memory, and the program code. This comprehensive encryption approach effectively protects data, computations, and code from potential attacks, even when adversaries have root access.

Specifically tailored as a secure container environment for Docker, SCONE empowers the execution of Linux applications within secure containers, harnessing the capabilities of SGX. This integration reinforces security measures and protects sensitive data from unauthorised access and modification. SCONE's versatility extends to supporting a wide range of popular programming languages, including Python, Javascript, Java, C, C++, Rust, and Go, as well as some older languages like Fortran [129].

One of the primary advantages of SCONE is its inherent capability to accommodate a Small TCB via its architecture. To achieve this, SCONE presents a C standard library interface to container processes, ensuring implementation through static linking against a libc library. This approach significantly reduces the TCB's size, thereby minimizing potential attack surfaces and elevating the overall system's security.

Another critical aspect of SCONE is its unwavering commitment to maintaining low overhead for secure containers. To achieve this, SCONE employs a user-level threading implementation that strategically reduces costly enclave transitions of threads. SCONE effectively addresses possible performance limitations and promotes efficient processing inside the secure environment by optimising the duration of time threads spend within the enclave. Additionally, SCONE effectively maps OS threads to logical application threads within the enclave, optimizing the scheduling of OS threads when blocked due to thread synchronization, further enhancing the system's overall performance [14].

Furthermore, SCONE is thoughtfully designed to retain compatibility with Docker, ensuring secure containers operate seamlessly within the Docker engine. This means that developers can continue working within familiar Docker environments while benefiting from the added security of SCONE. The deployment of secure containers with SCONE requires only an SGX-capable Intel CPU, an SGX kernel driver, and, optionally, a kernel module for asynchronous system call support, streamlining the setup process for devel-

opers.

2.4.8 vSGX

SGX, once the dominant standard for TEE-based applications, has fostered a robust ecosystem with numerous open-source and commercial projects built upon it. However, a noteworthy constraint of SGX lies in its Instruction Set Architecture (ISA) extension, which imposes a rigorous demarcation between software applications, classifying them into trusted and untrusted components. The trusted software modules find their abode within secure enclave regions, accessible exclusively through the execution of code in enclave mode—a novel CPU operation mode. This paradigm compels developers to undertake the arduous task of either refactoring existing applications or constructing new ones, conforming rigorously to Intel's SGX software specifications. This predicament leads to a situation where SGX-developed applications find themselves tethered to SGX processors, resulting in a condition colloquially referred to as vendor lock-in [160].

In response to this conundrum, vSGX emerged as a pioneering system, promising binary code compatibility for SGX enclave software partitioned to enable direct execution on AMD processors. In conceptual terms, vSGX can be envisioned as an SGX hardware module seamlessly integrated into a SEV environment. The underlying notion revolves around harnessing the VM protection furnished by SEV while facilitating the execution of the trusted enclave of legacy SGX applications within a distinct VM. Despite the inherent disparities in design philosophy between SGX and SEV, vSGX effectively attains security assurances akin to SGX, ensuring the secure execution of SGX enclaves while preserving the protective advantages inherent to SEV [160].

At the core of vSGX's architecture lies the concept of transparently interposing the execution of enclave instructions to accommodate the SGX ISA extensions. This involves the consolidation of encrypted virtual memory segments originating from separate SEV virtual machines, resulting in the creation of a unified virtualized address space, reminiscent of the SGX environment. Additionally, vSGX offers a mechanism for attesting to the authenticity of the TEE and the integrity of enclave software through a trust chain anchored in the SEV hardware. Consequently, vSGX achieves a level of security assurance within the SEV framework that aligns with the robust security guarantees offered by Intel SGX [160].

2.4.9 Enarx

Enarx serves as an application deployment framework that empowers programs to run inside TEE without necessitating modifications for specific platforms or SDKs. The framework manages attestation and facilitates delivery into a run-time *Keep* based on We-

Containers	**API**	**Language(s)**	**Open Source**	**Hardware**	
				Intel SGX	**AMD SEV**
Apache Teaclave	WASI	C/C++, Python	✓	✓	
EGo SDK	libcwrapper	Go		✓	
Fortanix EDP	LibOS	Rust	✓	✓	
Mystikos (μkernel)	LibOS	C/C++	✓	✓	
Gramine	LibOS	C/C++	✓	✓	
Occlum	LibOS	C/C++, Python	✓	✓	
SCONE	libcwrapper	Rust, C/C++, Python, Javascript, Java, Go, Fortran	✓	✓	
vSGX	LibOS	Rust, C/C++	✓	✓	
Enarx	WASI	Rust, C/C++, Python, Haskell	✓	✓	✓

Table 2.1. *Trusted Containers Comparison*

bAssembly (Wasm), granting developers an extensive array of language options for implementing their systems. A notable advantage of Enarx as compared to other Trusted Containers is its independence from CPU architecture, allowing the same application code to be effortlessly deployed on various platforms. As a result, the need for cross-compilation and distinct attestation procedures across hardware manufacturers is eliminated.

In Table 2.1, we undertook a comparison between Enarx and a compilation of eight notable Trusted Containers that remained operational throughout the entirety of 2022. Enarx stands out due to its extensive selection of programming languages, its ability to seamlessly interface with a variety of hardware configurations, and its designation as an open-source solution. These combined factors position Enarx as an especially beneficial choice for the development of our privacy and anonymity solution, PESE. Moreover, its notable advantages in enhancing software development and fortifying security paradigms in contemporary computing contexts amplify the persuasiveness of this selection. Enarx is further explored and analysed in Chapter 3.

2.5 Privacy and Anonymity

Privacy and anonymity are two fundamental concepts that have become increasingly important in the digital age. As our lives become more intertwined with technology, we are constantly generating data that can be used to identify and track us. Privacy is the right to control one's personal information. This includes the right to decide who has access to this information, how it is used, and for what purposes. Anonymity is the state of being unknown or unidentifiable. The internet serves as an expansive repository of data, offering information on a wide range of topics. Individuals across different categories, classes, and countries rely on search engines to access the information they seek. WSEs play a vital role in this process by maintaining a query log, which records all submitted queries [146]. These logs contain various details, including the user's query content, machine IP address, operating system information, browser type, query timestamp, browser language, preferences, and potentially used cookies that uniquely identify the user's browser. Web search is unquestionably the most widely utilized online service, with Google alone handling over 3.5 billion queries daily [95]. Query logs hold significant value as they provide insights into individuals interests and activities, contributing to the vast pool of personal information amassed under the private sector's less stringent data collection and usage policies [28].

WSEs retrieve information from the vast expanse of data based on user-generated queries and utilize query logs to build user profiles and deliver personalized search results [33, 65]. To generate revenue, WSEs often sell these query logs to marketing companies [146]. However, the release of pseudonymized datasets of search queries, which inadvertently revealed certain users' identities based on personal information contained in their queries, sparked the AOL scandal [146]. Unfortunately, query logs are vulnerable to malicious attacks, with hackers seeking to sell the acquired personal information to other organizations. This poses a significant risk, as the information stored in query logs can expose sensitive details such as users' interests in future products, employment information, health concerns, and political or religious beliefs [133].

To address these privacy concerns, two methods are commonly employed: unlinkability and indistinguishability [41, 34, 94]. Unlinkability focuses on anonymizing users' identities through anonymous communication protocols like Tor [41], Dissent [34, 156], or RAC [94]. However, it has been demonstrated that even with anonymity protection, the content of search queries can still be used to re-identify users [106]. To overcome this limitation, the second approach aims to enforce indistinguishability between user profiles and queries. This approach aims to make it challenging for search engines to differentiate between real and fake queries generated on behalf of users. Examples of such solutions include TrackMeNot [6] and GooPIR [42]. However, research has shown that search engines can still distinguish between real and fake queries, thereby undermining the effectiveness

of these methods [110]. The following sections delves into the traditional privacy and anonymity solutions that serve as the foundation for the development of our privacy and anonymity solution, PESE.

2.6 Brief History of Privacy and Anonymity

The notion of privacy has remained a contentious and evolving topic throughout the course of human history. Its definition has continuously evolved, influenced by a variety of cultures and historical circumstances. In actuality, privacy has its origins in ancient societies, where it constituted an integral facet of human existence. There is a common argument that the very comprehension of ethics and moral judgment is closely intertwined with the concept of privacy, as individuals reflect upon their thoughts, actions, and ethical considerations in solitude.

Early legal documents acknowledged the significance of privacy. The Code of Hammurabi, one of the most ancient known legal codes dating back to ancient Mesopotamia, featured provisions against the intrusion into an individual's home, underscoring the sanctity of personal space [135]. Similarly, Roman law addressed matters of privacy by defining the boundaries of personal space and behaviour.

The perception of what constitutes private has always been shaped by the era, society, and individual perspectives. In ancient Greece, privacy was a matter of philosophical contemplation. The writings of Socrates and other Greek philosophers introduced the concept of distinguishing between the outer and inner spheres, emphasizing the importance of preserving one's inner thoughts and experiences [77]. Aristotle further contributed to this concept by differentiating between the public and private spheres of existence, acknowledging that the private sphere held a special significance [100].

In the exploration of the historical trajectory of privacy, an integral facet is the examination of perspectives embedded in religious scriptures, including the *Torah, Bible, and the Quran*. The evolution of privacy concepts has been profoundly influenced by the interplay of religious doctrines, which provide guidance for individuals in understanding the sanctity of personal space and boundaries.

Leviticus, the third book of the *Torah*, specifically *Chapter 19, verse 16*, underscores the importance of refraining from spreading harmful information about others. This precept promotes a sense of privacy and instills respect for the reputation and well-being of one's neighbors. The verse explicitly states, *"You shall not go about as a talebearer among your people; neither shalt you stand up against the blood of your neighbor: I am the LORD."* [4]

In the *Gospel of Mark*, the second book of the *New Testament* in the *Bible*, *Chapter 1, verse 35* recounts a significant moment: *"Very early in the morning, while it was still dark,*

[4] https://www.jewishvirtuallibrary.org/vayikra-leviticus-chapter-19

Jesus got up, left the house, and went off to a solitary place, where he prayed." [5] This early rising of *Jesus* and the deliberate choice to find a solitary place underscores the importance of personal, private moments dedicated to spiritual reflection and communication. In the second book of *Exodus* in the *Old Testament*, there are verses that allude to the significance of personal space and the inviolability of one's dwelling, notably *Chapter 20, verse 15* states: *"Thou shalt not steal"* [6] indirectly emphasizes the privacy of one's possessions.

Islamic perspectives on privacy are elucidated in the *Quran* through various verses and are further expounded in the *Hadiths* [7] of *Prophet Muhammad (PBUH)*. In *Surah-Noor, verse 27*, *Allah* instructs believers not to enter any house other than their own without permission and a greeting, promoting mindfulness and respect for others' privacy: *"O believers! Do not enter any house other than your own until you have asked for permission and greeted its occupants. This is best for you, so perhaps you will be mindful."* [8] Similarly, *Surah-Al-Hujurat, verse 12* underscores the importance of avoiding unwarranted intrusion into the lives of others, promoting a culture of trust and respect for personal boundaries. *"O believers! Avoid many suspicions, (for) indeed, some suspicions are sinful. And do not spy, nor backbite one another. Would any of you like to eat the flesh of their dead brother? You would despise that! And fear Allah. Surely Allah is (the) Accepter of Repentance, Most Merciful."* [9]

Prophet Muhammad (PBUH) explicitly forbade spying and prying into the confidential and personal affairs of others. He advises against suspicion, stating that it is the most dishonest form of communication. The guidance extends to discouraging inquisitiveness about one another and the act of spying. Hadith In *Sunan Abi Dawud* states, "*Avoid suspicion for suspicion is the most lying form of talk. Do not be inquisitive about one another or spy on one another."* [10]

These religious teachings have played a role in shaping societal norms surrounding privacy and emphasizing the ethical considerations associated with respecting the confidentiality and personal space of individuals. As the evolution of privacy continues, the influence of religious principles remains a significant aspect of the broader historical tapestry of privacy and anonymity.

In more recent history, privacy concepts have found their way into legislative instruments. The Finnish Constitution, for instance, does not explicitly define privacy, but it underscores the absolute nature of the privacy of letters, audio calls, and other confidential messages [3]. Similarly, The Fourth Amendment to the United States Constitution offers protections

[5] https://biblehub.com/mark/1-35.htm
[6] https://biblehub.com/exodus/20-15.htm
[7] Hadith or Ahadith (plural) are sayings and traditions of the Prophet of Islam Muhammed (PBUH).
[8] https://quran.com/en/an-nur/27
[9] https://quran.com/en/an-nur/27
[10] https://sunnah.com/abudawud:4917

against unreasonable searches and seizures, thereby imposing limitations on the right to privacy while emphasizing the importance of safeguarding individual liberties [4].

The essence of privacy as the *right to be left alone* was articulated by a Michigan Supreme Court Judge, highlighting the fundamental need for individuals to control their personal information and spaces [119]. This concept of privacy has been reinforced by modern legal frameworks, such as the General Data Protection Regulation (GDPR) of the European Union, which introduces the *right to be forgotten*, allowing individuals to request the removal of their personal data under certain circumstances [46].

On a global scale, the United Nations Universal Declaration of Human Rights in Article 12, and Article 17 of the International Covenant on Civil and Political Rights have enshrined privacy as a fundamental human right [149], underscoring the importance of protecting individuals from unwarranted intrusions into their personal lives and information.

Similar to Privacy, anonymity has a long and intricate history, dating back to ancient times. The term itself was borrowed into English from Greek in the late sixteenth century, often flaunting its scholarly origin through its spelling as *anonymos* [48]. One of the earliest instances of anonymity in antiquity is seen in the Greek playwright Aeschylus, who wrote under the pseudonym of his brother Euphorion, possibly to protect himself from repercussions for his critical works regarding the Athenian government [113]. Another example is found in the Roman satirist Juvenal, who employed pseudonyms to criticize the powerful and corrupt, thus shielding himself from censorship and punishment [152].

Anonymity has played a crucial role in historical political movements and revolutions. Anonymous pamphlets and letters were disseminated during the American Revolution, enabling individuals to voice their grievances and ideas without fear of reprisals from British authorities [132]. Similarly, the Federalist Papers, authored by Alexander Hamilton, James Madison, and John Jay under the pseudonym *Publius*, were instrumental in shaping the U.S. Constitution and the early American political landscape [64].

In the modern world, anonymity has gained increasing importance as a means of protecting privacy and freedom of expression. The advent of the internet has made it easier than ever for individuals to communicate and share information while remaining anonymous. It is also utilized by activists and dissidents to challenge governments and powerful institutions. For instance, the Anonymous hacker group has harnessed anonymity to carry out cyberattacks against governments and corporations [32].

Anonymity systems enable users to retain their anonymity and guarantee that their identity stays concealed from opponents who desire to find it. In today's society, where personal data is gathered and retained on a massive scale, the demand for privacy and anonymity has intensified. Different technologies have been developed to protect privacy and anonymity. These technologies are essential for protecting privacy and anonymity in

an era of mass surveillance and data collection.

2.7 Evolution of Privacy and Anonymity System

The first anonymity technologies were developed in the early 1990s. These technologies were designed to allow users to communicate with each other without revealing their IP addresses or other identifying information.

2.7.1 Type 0: Remailing System

The anon.penet.fi remailer was one of the first and popular anonymous remailing service [98]. Operated by Johan Helsingius under the pseudonym *Julf*, this system was active in Finland from 1993 to 1996 [38]. Its primary function was to provide users with anonymous email accounts by relaying messages between their genuine email addresses and assigned pseudonyms. Additionally, it removed identifying headers from remailed messages. The aforementioned system is often known as a type 0 anonymity service [59].

The impact of the anon.penet.fi remailer on the field of anonymity research was profound [108]. It represented one of the earliest instances of a functional online anonymity system, and its eventual closure sparked significant discussions about the privacy rights of anonymous users. The legal proceedings that led to the shutdown of the remailer also played a role in defining the legal framework surrounding systems for anonymous communication.

However, the anon.penet.fi remailer had its limitations. Firstly, it offered only rudimentary anonymity, as a passive observer with the capability to monitor internet traffic could effortlessly uncover the genuine email addresses associated with pseudonyms [59]. Furthermore, the remailer itself had the potential to compromise the identities of its users. This risk arose from the remailer maintaining a concealed identity table, linking real email addresses with pseudonyms. In the event of a security breach or a legal subpoena compelling the operator to reveal user identities, the shield of anonymity was at risk [38].

In 1996, Helsingius was summoned to testify in a legal case, which necessitated the disclosure of certain user identities. Initially, Helsingius declined to cooperate, driven by his commitment to protect user privacy. However, the court compelled his cooperation, resulting in the disclosure of the anon.penet.fi remailer, as it could no longer ensure user anonymity [38].

The anon.penet.fi remailer stands as a pioneering example of an anonymity system that had a substantial impact on online anonymity research. It effectively raised awareness about the importance of online anonymity and the intricacies involved in designing and implementing anonymous communication systems. Additionally, the disclosure of the remailer initiated a crucial dialogue concerning the legal rights and protections for anony-

mous users.

2.7.2 Type I: the Cypherpunk remailer

In the previous iteration of the Type 0 remailing system, two critical issues posed substantial challenges [98]. The first major concern centred around the system's susceptibility to traffic analysis, which enabled adversaries capable of monitoring traffic to effectively link messages. This vulnerability arose from the fact that both outgoing and incoming messages exhibited similar sizes and travelled within a short timeframe [98]. The second notable problem revolved around the retention of the connection between pseudonyms and real addresses on the remailing server. In the event of a compromise or legal mandate, the operator of the system's most sensitive component could unveil the pseudonymous identities of all recipients [98].

In response to these challenges, a more sophisticated version emerged, known as Type I remailers or Cypherphunk remailers, which was initially pioneered by notables Eric Hughes and Hal Finney [38]. These advanced remailers adopted a comprehensive approach to anonymity. These remailers employed a robust methodology to obfuscate all traceable information, employing private keys for decryption. The inaugural code-base was disseminated via the cypherpunks mailing list, a development that bestowed upon them their enduring moniker.

Encryption was executed through the application of Pretty Good Privacy (PGP) public key encryption functions, and the encoding process was designed to be manually executed, leveraging conventional text and email editing tools [38, 87]. Users were afforded the flexibility to create chains of multiple remailers, a strategic approach designed to mitigate reliance on any single remailer to preserve anonymity. The introduction of reply blocks offered a feature-rich experience, enabling users to communicate anonymously. In this setup, a user's email address was encrypted using the remailer's public key and embedded within a specialized header. Should a user desire to respond to an anonymous email, the remailer would decrypt it and facilitate the transmission of the response. Notably, Type I remailers outperformed their predecessors by providing enhanced protection against attacks. They did not maintain a database associating real user identities with pseudonyms, and crucial addressing information necessary for responding to messages was encapsulated within the messages themselves in an encrypted format [98].

The encryption protocols employed during the network traversal of messages effectively thwarted basic passive attacks rooted in the observation of bit patterns in incoming messages and their correlation with outgoing ones. However, there remained a minimal leakage of information, primarily related to message size. PGP, despite its role in message compression, did not actively conceal message size, making it feasible for observant actors to trace messages in the network by simply noting their length [98]. Furthermore, the

reply blocks introduced an element of vulnerability, as they could be reused. Malicious actors could encode numerous messages, facilitating a statistical attack to determine their ultimate destination. This attack vector could be employed for multiple hops. Despite these acknowledged limitations, Type I remailers gained widespread acceptance due to their reply capabilities, which enabled the utilization of Nym Servers [11].

The persistent existence of Type I remailers has raised concerns within the anonymity research community, as they are challenging to phase out. The inherent security drawback of the reply block feature, which is absent in later Type II Mixmaster software, is twofold: it is essential for the construction of Nym Servers but remains vulnerable even to passive adversaries [38]. These concerns have led to the subsequent development of Type III remailers, as detailed in the subsequent section. These remailers are designed to provide an elevated level of resistance to traffic analysis and offer secure single-use reply blocks, presenting a promising path forward in enhancing privacy and security within the realm of remailing systems.

2.7.3 Type II and III: Mixmaster remailer

In the mid-1990s, a significant advancement occurred in the field of remailer technology, resulting in the creation of Type II and Type III Mixmaster remailers. These were designed to enhance protection against traffic analysis, a critical concern for maintaining online communication anonymity [20]. These new versions of remailers brought about several improvements to the existing Type I remailer technology [38].

Type II remailers implemented four primary strategies to improve user protection [59]. It introduces the concept of connecting multiple remailers in a chain with encryption between them. This effectively concealed the origins and destinations of messages, making it difficult for adversaries to trace the communication path. To counter passive traffic monitoring, Type II remailers kept a consistent message length, preventing attackers from identifying messages based on their size and thus protecting user privacy. These remailers operated in a stateful manner, refraining from resending messages to prevent replay attacks. This strategic choice ensured that adversaries couldn't intercept and resend messages repeatedly to determine their intended recipients. Additionally, Type II remailers utilized message reordering to enhance security against eavesdropping, adding complexity for potential attackers [98].

The evolution of remailer technology continued with the introduction of Type III Mixminion anonymous remailers, which offered substantial enhancements to user privacy and security [59]. They provided anonymous reply addresses, allowing users to respond to messages while maintaining their privacy. Type III Mixminion remailers adopted TLS-

[11] https://www.whonix.org/wiki/Nymservers

encrypted Simple Mail Transfer Protocol (SMTP) with unique encryption keys for each message, ensuring the confidentiality and integrity of email communications [20]. They introduced exit policies, allowing Mixminion node operators to establish rules for filtering out abusive email messages, thus promoting the responsible operation of the remailer network. To further mitigate vulnerabilities related to traffic analysis, Type III Mixminion remailers included dummy noise traffic to confuse attackers trying to discern communication patterns [37].

It's important to emphasize that while email remailers, including Mixmaster and Mixminion, provide a high degree of privacy and anonymity, they cannot fully address the inherent privacy limitations in the underlying TCP/IP stack. Nevertheless, the protective methods and principles introduced in these remailer systems have influenced more comprehensive anonymity solutions, such as the Tor and I2P networks, which offer more robust support for anonymous TCP/IP communication.

2.8 Modern Privacy and Anonymity System

While numerous contemporary privacy and anonymity solutions have been proposed and are currently accessible, this thesis delves into the discussion of the most prevalent services described below, which effectively ensure anonymity within the modern era.

2.8.1 Tor

In 1996, the concept of Onion Routing was introduced as an evolution of mix remailing systems [60], and it later evolved into a practical low-latency communication system [61]. It took several years to fully implement the routing network. In 2004, Syverson, Dingledine, and Mathewson presented "Tor: The Second-Generation Onion Router" along with the source code of The Onion Routing (Tor) [41]. This marked the beginning of Tor's provision of anonymous TCP/IP connections for users.

Tor is a widely-used low-latency anonymity network that conceals the user's original IP address [26]. It serves a wide range of purposes, both legal and illegal, attracting users from various backgrounds, including regular citizens concerned about their privacy, corporations seeking to protect sensitive information, law enforcement agencies conducting covert operations, human rights activists, and journalists needing secure communication.

To ensure online privacy and anonymity, Tor employs several key techniques. Figure 2.5 illustrates a simplified representation of the Tor Network. First and foremost, users do not establish direct connections to their intended destinations. Instead, they connect to the Tor network, which acts as an intermediary and forwards their connection to the ultimate destination. Secondly, the destination itself remains oblivious to the user's original IP address. Rather, it only perceives the IP address of a Tor relay, ensuring that the user's

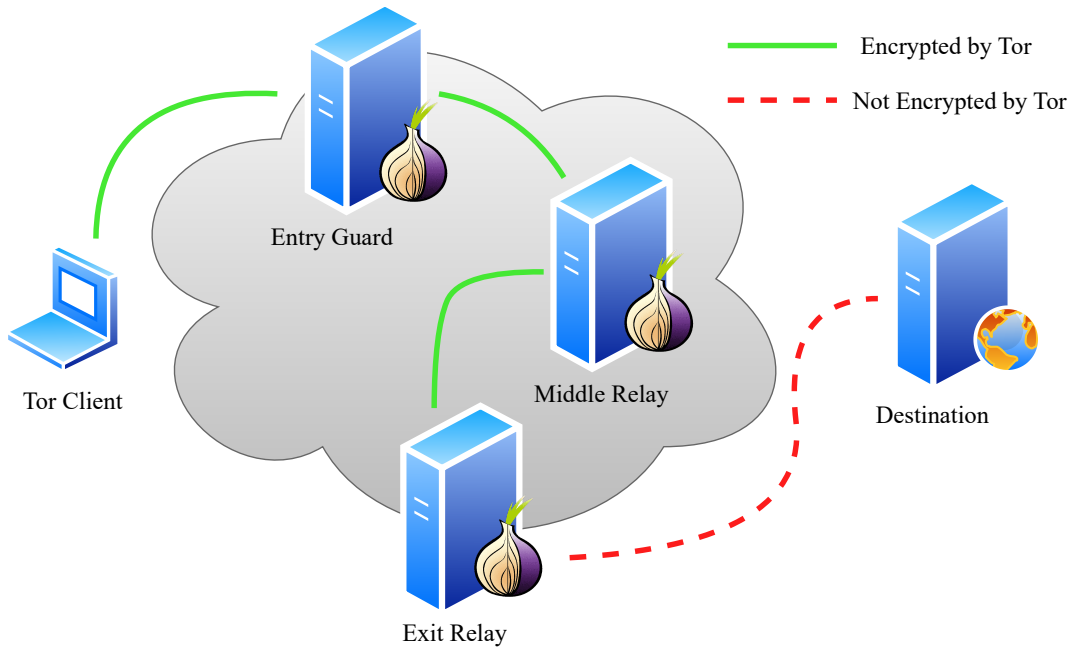

Figure 2.5. *Simplified Tor Network*

identity remains concealed. The third key principle involves the establishment of a three-relay circuit within Tor. Each relay within this circuit possesses distinct knowledge, thereby contributing to the overall anonymity of the user. The first relay possesses knowledge of the user's IP address but remains unaware of the final destination. The second relay, positioned between the first and third relays, exclusively observes incoming traffic from the first relay and relays it to the third relay. Meanwhile, the third relay holds knowledge of the destination but remains uninformed about the user's original IP address. This relay architecture is designed to compartmentalize information and enhance user anonymity. Fourthly, Tor implements a multilayered encryption approach, akin to the layers of an onion. As user traffic traverses each of the three relays, it is successively encrypted and encapsulated in additional layers. This encryption strategy is instrumental in safeguarding the confidentiality of user data and is, in fact, the origin of the term *onion routing*. Furthermore, it is pertinent to note that Tor operates as an open and freely accessible network. This characteristic attracts a significant user base and numerous volunteer-operated relays. While this openness fosters diversity and inclusivity, it also presents challenges in terms of conducting global traffic and timing analysis against the network. The sheer volume of users and relays within the Tor network renders such analysis intricate and demanding, contributing to the network's robustness in preserving user privacy [98].

The Tor circuit ensures end-to-end encryption using the 128-bit Advanced Encryption Standard (AES) cipher in counter mode, along with checksums for integrity verification [98]. The network comprises a distributed system of volunteer-operated relays, including entry nodes (guards), middle nodes, and exit nodes. These relays randomize the route of

user traffic, adding an extra layer of anonymity and thwarting attempts to trace user paths.

To counter traffic analysis, Tor employs a *padding technique*, which involves adding dummy data to transmitted packets, making it difficult to distinguish real traffic from encrypted Tor traffic. Nevertheless, despite its effectiveness, Tor is not entirely invulnerable. As a low-latency anonymity network, it remains susceptible to traffic correlation attacks by adversaries eavesdropping on both ends, a limitation inherent to such a network [131].

2.8.2 I2P

In 2002, the Invisible Internet Project (I2P) emerged as a pioneering and robust privacy-centric network, dedicated to facilitating anonymous communication and ensuring secure data transfer [71]. I2P inception can be traced back to the pressing need for a decentralized, self-sustaining network that prioritizes user privacy and security.

The project was conceived by a dedicated group of developers who were fueled by the vision of a darknet, where users could communicate and exchange data without the looming spectre of surveillance or interference. This ambitious initiative carved out a niche for itself by providing a secure and private platform that caters to a diverse user base, encompassing individuals with privacy concerns, as well as activists and journalists operating in regions where the freedom of expression hangs in the balance.

I2P shares a fundamental objective with its predecessor Tor, but it sets itself apart through a distinctive approach to safeguarding online privacy. At the core of it's design lies a network of interconnected peers, with each user participating by running an I2P router. These routers collaborate to form a mesh network, wherein data is meticulously encrypted and then routed through multiple nodes, effectively obscuring both the source and destination of the traffic.

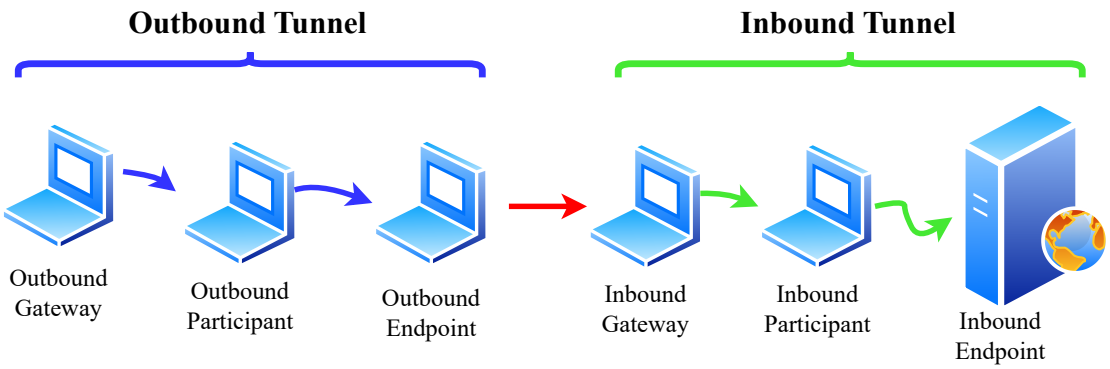

Figure 2.6. *Simplified I2P Network*

Figure 2.6 illustrates a simplified representation of the I2P Network. Central to I2P lies the concept of *tunnelling*, which introduces two crucial components: outbound tunnels and inbound tunnels.

Outbound tunnels are responsible for guiding messages away from the tunnel creator, while inbound tunnels bring them back to the source. In this system, the sender or client initiates an outbound tunnel, adding directives to her encrypted messages. The endpoint of the outbound tunnel decrypts the message and contains instructions to forward it to the correct inbound gateway server. To facilitate end-to-end communications between clients and servers, I2P utilizes *garlic encryption*, which bundles multiple messages into a single *garlic message* on the client router [7]. This process, although similar to onion routing, has its distinct characteristics. By encrypting messages with a specific public key, intermediary peers remain unaware of the client/server relationship and the ultimate message destination. It's crucial to note that I2P's mechanism operates on a message-based system, where data loss can potentially occur during transmission [7].

Despite its commendable strengths in ensuring anonymity and privacy, I2P faces several challenges and limitations. One notable issue is its comparatively small user base and network size when compared to the open and widely accessible Tor network. The limited number of users and relays in I2P may render it more vulnerable to determined adversaries who could engage in traffic analysis. Additionally, accessing regular internet resources through I2P can be hindered by increased latency introduced by routing through multiple peers.

2.8.3 Tor vs I2P Comparison

Comparing Tor and I2P is essential for understanding two prominent anonymity networks that play a crucial role in online privacy and security. Both Tor and I2P serve as powerful tools for anonymizing internet traffic and protecting users from surveillance, censorship, and potential threats to their digital identities.

Tor, with its onion-routing mechanism, creates a multi-layered encryption process that bounces data through a series of volunteer-operated servers, making it effective for achieving anonymity but potentially slower due to the increased routing. On the other hand, I2P utilizes a distributed peer-to-peer network, focusing on hidden services within the network itself. This approach can result in faster connections but may require users to adapt to its unique ecosystem.

While these networks share a common goal of enhancing online anonymity, they employ different architectures, protocols, and methods to achieve this objective as we discussed in above Subsection 2.8.1 and Subsection 2.8.2. In the following Table 2.2, we explore and compare the features of Tor and I2P according to the research conducted by Ali et.al

[7], offering valuable insights into their respective capabilities.

Features	**Tor**	**I2P**
User Base	*Much Bigger*	*Small*
Visibility in Security Community	*More visible*	*Less visible*
Documentation	*Well documented*	*Less documented*
Number of exit nodes	*Significant*	*Insignificant*
Memory usage	*More efficient*	*Inefficient*
Bandwidth overhead	*Very low*	*Very high*
Centralized/Distributed control	*Centralized*	*Distributed*
Latency	*Low*	*High*
Language	*C*	*Java*
Nodes selection criteria	*Trusting claimed capacity*	*Continuously profiling and ranking performance*
Packet/Circuit switched	*Circuit switched*	*Packet switched*
Uni/Bi Directional Circuit	*Bi directional*	*Uni directional*
Protecting client activity	*Less protected*	*Much protected*
Life of tunnels/circuits	*Long lives*	*Short lives*
TCP/UDP Transport	*TCP*	*Both TCP/UDP*

Table 2.2. *Comparison of Tor and I2P*

2.8.4 Bitcoin Mixers

Following the emergence of cryptocurrency in 2009, a new era of digital finance was ushered in by the enigmatic figure known as Satoshi Nakamoto [111]. Nakamoto's brainchild, Bitcoin, disrupted the traditional banking system by establishing a decentralized, peer-to-peer network for financial transactions. While this innovation empowered individuals with unprecedented control over their finances, it also raised concerns about transaction privacy. The inherent anonymity of cryptocurrency transactions, designed to protect users' identities, led to worries about potential misuse, such as money laundering and criminal financing. As a result, this paved the way for the development of innovative tools called Bitcoin mixers, which were designed to enhance the privacy of cryptocurrency transactions.

The concept of Bitcoin mixers originated in the early days of cryptocurrency, driven by the

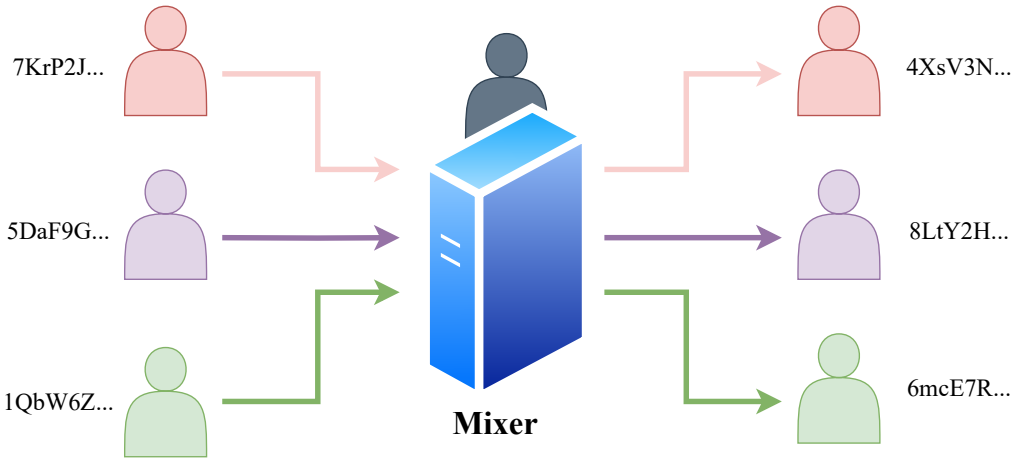

Figure 2.7. *High level illustration of Bitcoin mixer with three participants and a central operator. Participants send Bitcoin, and the mixer redistributes it to new addresses, preventing tracking of their initial deposit [103].*

necessity to safeguard user privacy and anonymity in light of blockchain's inherent transparency. Figure 2.7 illustrates the overall operational characteristics of a mixer with three users [103]. A mixer's primary function involves sending users' Bitcoins to the mixer's wallet, along with designated destination addresses for the mixed Bitcoins. The mixer then combines these funds with those of other users and distributes them to the specified destination addresses, effectively breaking the connection between the sender and receiver. Although the high-level process may seem traceable, mixers employ obfuscation techniques that make it challenging to trace transactions and identify the use of mixers on the blockchain.

One of the earliest documented Bitcoin mixers, known as TumbleBit, emerged in 2013 and used a centralized server to aggregate Bitcoin deposits from multiple users and distribute them to their intended destinations [103]. Over time, various Bitcoin mixers with different privacy-enhancing techniques have appeared. Some rely on centralized servers, while others use decentralized protocols to ensure trust. This evolution in Bitcoin mixers reflects the growing demand for privacy solutions in the cryptocurrency space. A study conducted by Pakki et al. [103] evaluated various features of Bitcoin mixing services as shown in Table 2.3. The research categorizing them as Trusted or Untrusted based on their activities and standing on the Bitcointalk forum [103]. A ✓ (checkmark) indicates that the service offers a specific feature, while a ✗ (cross) indicates the absence of that feature. Fields marked with a - (dash) were either not found or not applicable to the service. Trusted mixers maintained consistent communication with an active user base on the forum and had no record of scam accusations at the time of the study, whereas Untrusted mixers had poor communication with their users and had one or more scam accusations

[103].

Mixer	Year	Mixing Fees	Account	Dist. Control	Delay	Multi Output Addr	Multi Input Addr	Tor	Clearnet	Bitcointalk	Open Source	Min. Blocks	Forum Posts	Scam Claims	Trusted
Samurai Whirlpool	*2015*	✓	✗	✓	✗	✓	✗	✓	✓	✗	✓	-	-	-	✓
Crypto Mixer	*2016*	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗	1	356	✗	✓
Mixer.money	*2016*	✓	✗	✗	✓	✓	✗	✓	✓	✓	✗	-	151	✗	✓
BitCloak	*2016*	✓	✗	✗	✓	✓	✗	✓	✓	✓	✗	1	174	✗	✓
ChipMixer	*2017*	✗	✗	✓	✓	✓	✗	✓	✓	✓	✗	1	1887	✗	✓
BitMix.biz	*2017*	✓	✗	✗	✓	✓	✗	✓	✓	✓	✗	1	147	✗	✓
FoxMixer	*2017*	✓	✗	✓	✓	✓	✗	✓	✓	✓	✗	6	39	✗	✓
Wasabi Wallet	*2018*	✓	✗	✗	✗	✓	✗	✓	✓	✗	✓	-	-	-	✓
MixTum	*2018*	✓	✗	✗	✓	✓	✗	✓	✓	✓	✗	1	99	✗	✓
Bitcoin Mixer	*2019*	✓	✗	✓	✓	✓	✗	✓	✓	✓	✗	1	108	✗	✓
Sudoku Wallet	*2019*	✓	✗	✗	✓	✓	✗	✓	✓	✓	✗	3	68	✗	✓
Bitcoin Fog	*2011*	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	6	647	✓	✗
PenguinMixer	*2017*	✓	✗	✗	✓	✓	✗	✓	✗	✗	✓	2	-	-	✗
Blender.io	*2017*	✓	✗	✗	✓	✓	✗	✓	✓	✓	✗	3	103	✓	✗
BMC Mixer	*2017*	✓	✗	✗	✓	✓	✓	✓	✗	✓	✗	2	2	✗	✗
SmartMix	*2019*	✓	✗	✓	✓	✓	✗	✓	✓	✓	✗	3	170	✓	✗
Mixer Tumbler	*2019*	✓	✗	✗	✓	✗	✗	✓	✓	✓	✗	3	17	✗	✗
AtoB Mixer	*2019*	✓	✗	✗	-	✓	✗	✓	✓	✓	✗	-	102	✓	✗
Anonymix	*2020*	✓	✗	✓	✓	✓	✗	✓	✓	✗	✗	1	-	-	✗
BlockMixer	*2020*	✓	✗	-	-	✗	✗	✓	✓	✓	✗	3	1	✗	✗
DarkWeb Mixer	-	✓	✗	✗	-	✓	✗	✓	✗	✗	✗	-	-	-	✗

Table 2.3. *Illustration of different Bitcoin Mixer Functions within Existing popular Bitcoin Mixing Services by the year 2020 [103]*

The analysis revealed that 11 mixers were classified as Trusted, while 10 fell into the Untrusted category [103]. The inclusion of the table in the present thesis serves the purpose of providing the reader with an understanding of the popularity and development of anonymity services throughout various years. Despite the presence of current bitcoin mixing services, the emergence of several bitcoin mixing services indicates a shared desire among developers and consumers to enhance their levels of privacy and anonymity. The Bitcoin mixing ecosystem attracts a diverse range of users, many of whom simply want to maintain their anonymity. The association of scams and sub-optimal implementation by some mixing services has led to the introduction of secure protocols in academic literature. These proposed solutions aim to ensure accountability among mixing services

and establish secure communication between participants while preventing the leakage of input and output permutations. A discussion of Bitcoin mixers is crucial, as it serves as one of the foundational pillars for the development of our Privacy and Anonymity Solution PESE.

3. THE ENARX FRAMEWORK

Enarx, a term originating from Latin, can be interpreted as *"within the citadel"* or *"inside the stronghold,"* and it was coined by its founders, Nathaniel McCallum and Mike Bursell [45]. It is a project under the Linux Foundation's Confidential Computing Consortium, has been an integral part since its inception on 31st October 2019 [45]. A significant benefit of Enarx lies in its CPU-architecture independence, which enables the deployment of the same application code across multiple targets while abstracting complexities like cross-compilation and varying attestation methods among hardware vendors. In the subsequent sections, we will explore its technical features in detail.

3.1 Design Principles

When executing a workload on a host or system, whether situated in the cloud or on-premises, a complex hierarchical structure is evident, as illustrated in Figure 3.1. This structure, characteristic of both traditional cloud virtualization and container architecture, consists of diverse layers distinguished by various colours, symbolizing entities claiming ownership over specific layers or sets of layers. These entities encompass a wide range, from hardware vendors, Original Equipment Manufacturer (OEMs), and Cloud Service Providers (CSPs) to middleware vendors, Operating System vendors, application vendors, and the workload owner. The composition of layers may vary for each workload instance, and even when identical, differences may arise in version specifics, such as BIOS version, bootloader, kernel version, and other elements.

While the intricacies of layer composition and version specifics may be inconsequential in many cases, especially when CSPs shield users from such details. From a security standpoint, the concern extends beyond recognizing diversity in layer versions; it encompasses the multitude of elements and entities requiring trust for the secure execution of sensitive workloads on these platforms. Each layer and its respective owner must not only gain trust in fulfilling their functions but also in remaining uncompromised. Entrusting such a diverse array of entities poses a significant challenge when addressing the security implications of running sensitive workloads.

Enarx is founded on a set of fundamental design principles that are integral to its success [45]. These principles encompass various critical factors, each of which contributes to the

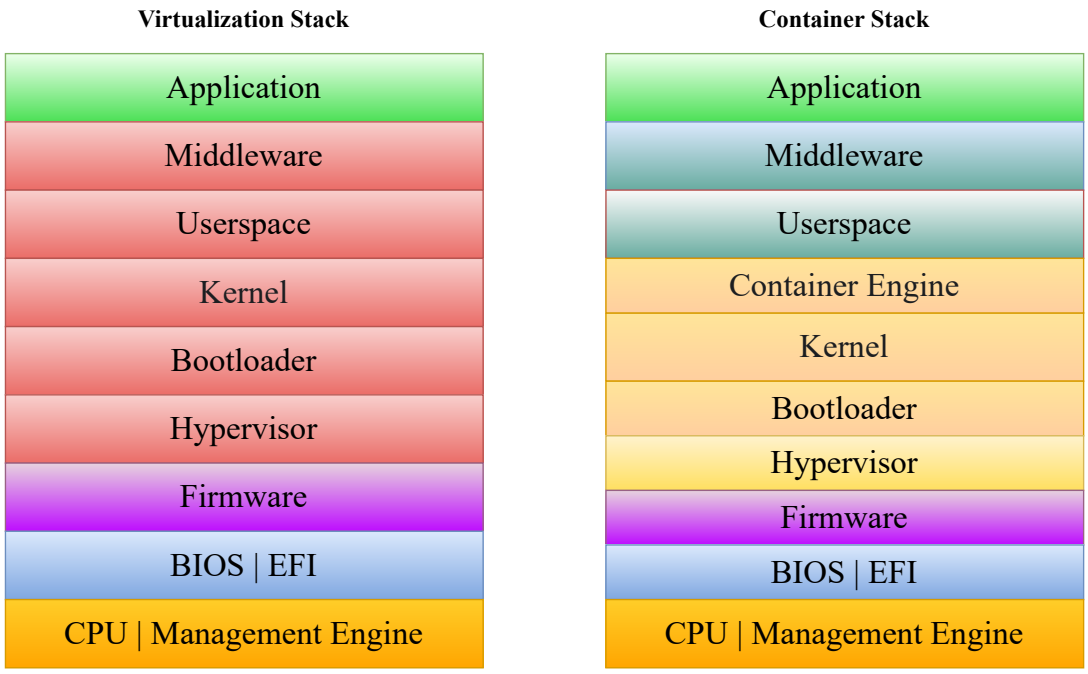

Figure 3.1. *Illustration of Virtualization and Container Stack*

overall security and functionality of the system.

When evaluating system security, the first step is to identify the TCB, which comprises components necessary to achieve the desired security level [52]. Enarx places significant emphasis on minimizing the TCB's size and subjecting trusted code to code-owner validation to mitigate potential attack vectors that could compromise the tenant's code and data. Furthermore, Enarx addresses vulnerabilities by keeping the network stack outside the TCB [45], as network stacks tend to be large, complex, and susceptible to privilege escalation and compromise [153].

Another key principle in Enarx is the notion of Minimum trust relationships. This principle is thoroughly discussed in Subsection 3.1.1, and it highlights the importance of establishing minimal trust dependencies to enhance overall security.

Deployment-time portability is yet another essential consideration for Enarx. By supporting multiple hardware architectures, Enarx enables its programs to be redeployed on various CPU architectures without the need for recompilation [45]. This flexibility enhances adaptability and ease of use.

Enarx's design also places a strong focus on auditability. By structuring the code into small, independent components that are easily comprehensible, the project fosters an environment conducive to thorough auditing. Avoiding run-time modularity of the basic platform further simplifies the auditing process [45]. The project is released under the Apache 2.0 open-source license [141], granting users the freedom to modify the program according to their needs and ensuring that auditability is accessible to all, not just a select

few.

Memory safety is another important design consideration for Enarx, which is accomplished predominantly through the use of the Rust programming language [45]. Rust's focus on memory safety reduces the dangers associated with memory corruption, thereby improving the system's overall stability and security [101].

Enarx's design principles encompass trust reduction, portability, data security, auditability, openness, and memory safety, all of which collectively contribute to the project's strong foundation and success.

3.1.1 Trust Architecture

Enarx is focused on addressing a crucial concern surrounding the trust relationships between different layers in the stack. The Enarx runtime stack is structured with four layers, ordered from the lowest to the highest level [45]:

1. ***VMM (Virtual Memory Manager)*:** The virtual memory manager serves as an intermediary between the processor and the memory system. Its purpose is to translate the virtual addresses of program instructions into corresponding physical addresses. In lieu of explicitly utilising physical memory addresses, the programme relies on this translation procedure. The VMM has the ability to map a particular virtual address to various physical memory addresses as needed. This flexibility enables it to rearrange data in the memory system without requiring application programme modifications [123].
2. ***(μkernel)* Microkernel:** The microkernel is a minimal operating system architecture that focuses on keeping the kernel's core functions basic, enhancing security, reliability, and maintainability. It achieves this by moving services into user-space processes, that communicate with the μkernel through controlled mechanisms. The design's advantage lies in isolating services, reducing the impact of failures, and enabling easy extensibility, making it suitable for embedded systems, real-time systems, and safety-critical applications [140].
3. ***Wasm (WebAssembly) runtime:*** The WebAssembly runtime serves as the framework to execute Wasm within a self-contained environment, enabling efficient and secure execution of Wasm modules [**Wasmtime**]. The Wasm runtime parses the Wasm bytecode and converts it into machine code compatible with the host platform for execution [151].
4. ***WASI (WebAssembly System Interface) implementation:*** WASI is a system interface created by the Wasmtime project specifically for Wasm. It aims to standardize the way Wasm interacts with OS-like functions such as files, sockets, clocks, and random numbers. Additionally, it incorporates capability-based security mea-

sures to ensure enhanced sandboxing and secure I/O operations. The primary goal of WASI is to provide a consistent and standardized interface for Wasm, enabling code portability and compatibility across different platforms and environments, thus allowing for secure execution in various contexts beyond web browsers [143].

At runtime, each of these layers undergoes cryptographic measurement and verification before being deployed. However, it is essential to understand that these four layers do not represent the entire stack. Additional components, including the CPU (and its firmware), the host kernel, and the application itself, also play crucial roles. Additionally, it is noteworthy that Enarx does not employ a TPM (Trusted Platform Module) as part of its trust architecture [45].

Figure 3.2 from the Enarx documentation [45], presents a concise representation of the layers and their roles in the VM case (AMD SEV).

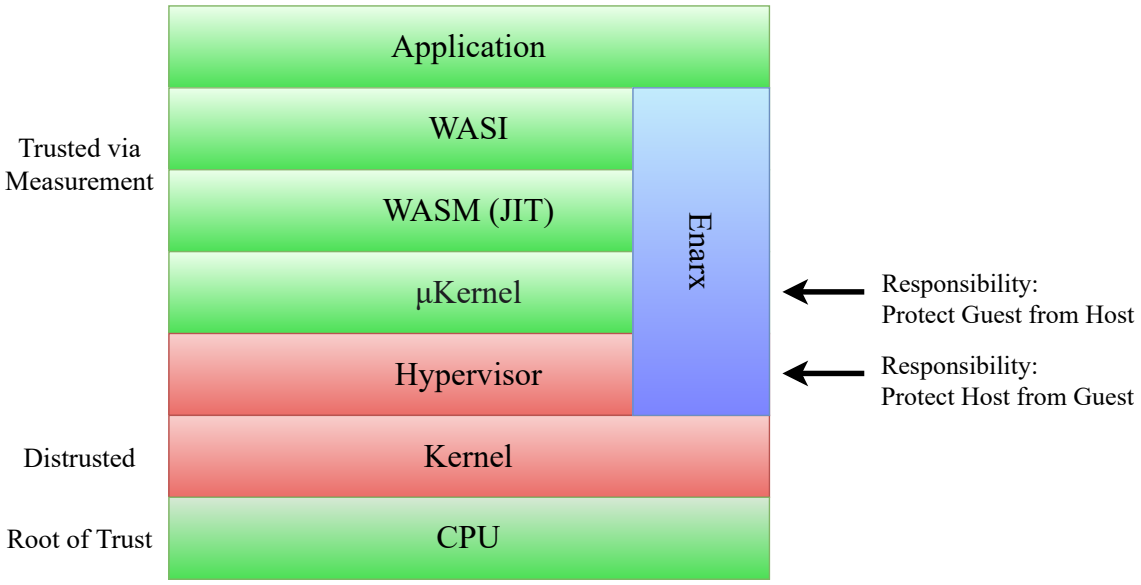

Figure 3.2. *Illustration of the Basic Layer Diagram for AMD SEV*

In the discussion, the trustor, which is the client or tenant intending to execute a workload, is depicted. In the Enarx architecture, this entity is referred to as the application. A foundational principle of Enarx is that the application should not place trust in the host, its owner, or its operator. While the host's owner or operator may have their own trust requirements, this discussion does not address those concerns [45].

Table 3.1 demonstrates the trust status of various components. For standard computation, the CPU must be trusted for executing operations. Enarx adopts a robust trust model, cryptographically verifying the CPU and firmware at runtime, acting as the hardware root of trust for the system's architecture. If any of these components are found invalid, Enarx

Components	Status	
	Trusted	Not Trusted
Central Processing Unit (CPU)	✓	
Kernel		✓
Virtualization Memory Manager (VMM)		✓
Microkernel (μkernel)	✓	
WebAssembly (Wasm)	✓	
WebAssembly Interface (WASI)	✓	
Application	✓	

Table 3.1. *Components and Enarx Trust Status*

prevents application scheduling on the host. The host's kernel is provided by the host, and the client deems it untrusted. Enarx supplies an untrusted VMM for memory management but limits access to encrypted memory pages. In contrast, Enarx's μkernel is trusted for standard kernel operations. The Wasm runtime, another trusted element, provides the runtime environment for applications within Enarx Keep, optimizing performance through silicon architecture-specific JIT compilation. Similarly, the WASI layer serves as a trusted interface for secure and portable Wasm applications on server-type systems. Lastly, the client-provided application layer, being trusted, represents the workload to run within Enarx Keep [45].

3.1.2 Threat Model

Enarx's threat model is built upon core principles of trust, outlined in the previous section in Table 3.1. These principles aim to create a secure environment for exchanging computing resources between two parties who lack mutual trust.

1. ***Host Perspective:*** The Host aims to offer its computing resources to the Guest in exchange for the value from the Guest's service. However, due to a lack of trust, the Host needs to ensure that the Guest cannot modify or tamper with the hardware, firmware, OS, or other software. The Guest must also be prevented from inspecting the execution context to avoid privilege escalation attacks [92]. Despite the mistrust, the Host still needs to maintain full control over resource allocation, allowing measurement, restriction, or termination of the Guest's resource usage at any time [45].
2. ***Guest Perspective:*** The Guest desires access to the Host's computing resources, but trust issues prevent this. To comply with data privacy policies and regulations, the Guest must protect the confidentiality and integrity of its data and code. It must

also prevent the Host from launching statistical analysis attacks [27, 120]. Therefore, the Guest requires robust protection from the Host for both its code and data. In addition, the Guest requires assurance that the Host will provide these protections irrevocably due to concerns regarding the Host's possible malice or compromise. Only relying on trust or legal mechanisms is insufficient and expensive [45].

In order to effectively address these concerns, it is imperative to implement an advanced resolution that encompasses hardware endowed with the capability to enforce the desired protections. Additionally, the incorporation of cryptographic attestation is indispensable to ascertain that the hardware appropriately governs the execution context [45]. This approach establishes a higher level of security and trustworthiness, forming a solid foundation for securely exchanging computing resources between the Host and the Guest within the Enarx environment.

3.2 Requirements

Enarx's primary objective is to offer a cross-platform runtime environment that ensures the secure execution of sensitive workloads. It achieves this by employing a platform-agnostic approach, making it adaptable to a wide range of systems, be it servers, desktops, or even embedded devices. In contrast to other SDKs, Enarx functions not solely as a development framework but also as a deployment framework.

One of Enarx's key components is its Wasm runtime 3, built upon Wasmtime. Rust is the primary language of Enarx, however, any language that can compile code into a Wasm binary is compatible with Enarx. This choice allows developers to use multiple programming languages, such as Rust, C, C++, C#, Go, Java, Python, and Haskell, for their implementations [45].

Enarx has minimal system library dependencies, including glibc [1], OpenSSL [2], and its own runtime. These libraries and utilities enable Enarx to effectively run and manage Keeps, handling essential tasks such as cryptographic operations, secure communication, and resource allocation.

Enarx is designed to be independent of CPU architectures, ensuring that the same application code can be deployed across various targets. This abstraction minimizes concerns related to cross-compilation and varying attestation mechanisms among different hardware vendors. Consequently, Enarx offers seamless execution on different silicon architectures. However, to run Keeps (secure enclaves) within a TEE instance, Enarx requires specific hardware equipped with TEE support. This hardware capability enables Enarx to create secure enclaves to protect sensitive data and code from the underlying

[1] https://www.gnu.org/software/libc/
[2] https://www.openssl.org/

system. Enarx currently supports Intel platforms like SGX [72] or TDX [9], AMD platforms like SEV [8], and upcoming platforms like Arms Realms and IBM's PEF without requiring recompilation of application code [45].

For development purposes, Enarx provides the *nil* backend, which does not rely on special hardware. This backend supports any of the architectures compatible with the Wasmtime crate. Additionally, for situations where accessing hardware with Intel SGX [72] or AMD SEV-SNP [8] support is not feasible, Enarx accommodates KVM as an alternative. KVM, a full virtualization solution for Linux on x86 hardware with virtualization extensions, facilitates testing on more commonly available hardware with virtualization support [45].

Enarx's security foundation lies in leveraging trusted hardware, hardware-based virtualization, and modern Linux features. This strategic combination enables Enarx to establish a secure platform, ensuring the execution of trusted applications in diverse environments.

3.3 Core Components

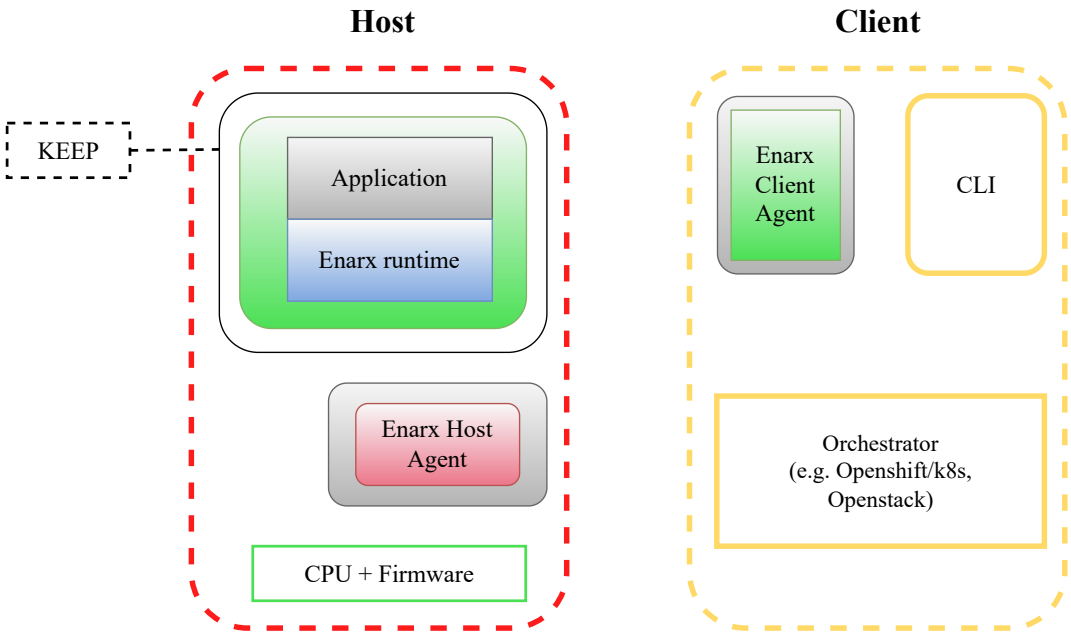

Figure 3.3. *Simplified representation of Enarx Architectural Components and Integration*

Enarx modules have been intentionally designed to exhibit modularity and interoperability, enabling their integration with a range of TEEs and hardware platforms. This adaptability extends not only to the TEEs themselves but also encompasses diverse programming languages, provided they can be compiled into Wasm. The Enarx schematic, as depicted in Figure 3.3 [24], visually elucidates the constituents of Enarx and their interconnections.

The hardware foundation consists of the CPU, kernel, and VMM. Stacked atop these layers are the Enarx components. The Enarx API and the central component engage with

the attestation element to validate the TEE's identity and integrity. Subsequently, the application is loaded into the TEE and launched. Execution within the TEE is facilitated by the Enarx runtime, leveraging the Wasm runtime environment. The management component supplies tools to oversee Enarx Keeps. A more intricate illustration of the Enarx components and their intricate interactions can be found in Figure 3.4 from the official Enarx documentation [45].

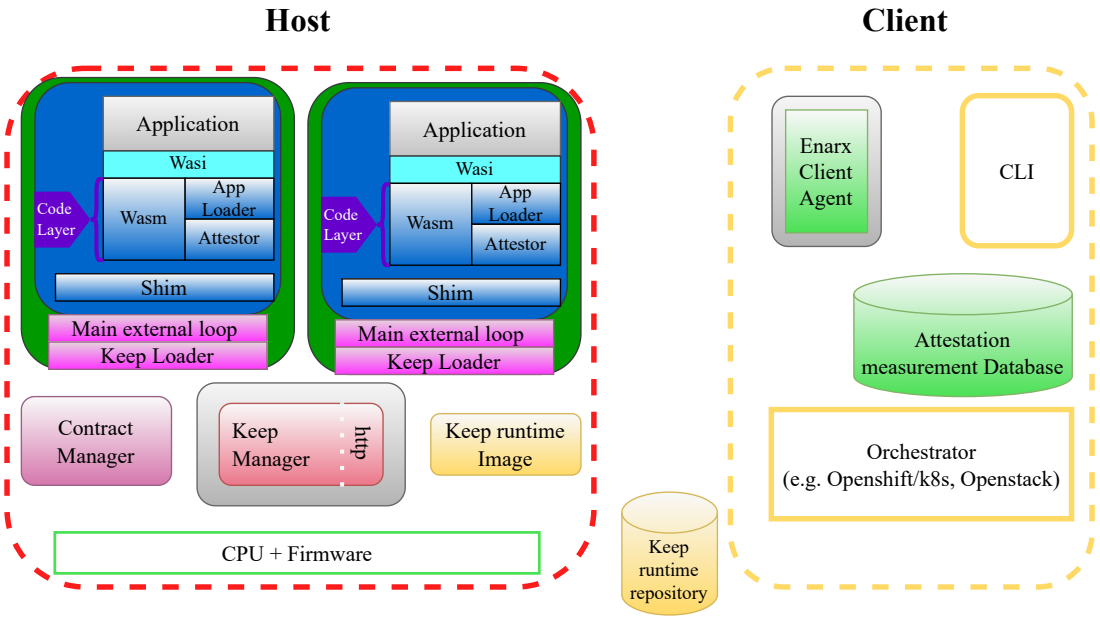

Figure 3.4. *Representation of Enarx Components and Integration*

3.3.1 Attestation:

Verification assumes a critical role within Enarx's framework, with Attestation serving as an essential core element. Its role encompasses confirming the host system's identity and integrity before deploying an application within a TEE. An application designated for execution within an Enarx Keep must undergo two assertions [45]:

1. Confirming the hardware TEE responsible for Keeps.
2. Evaluating the measurement of the Enarx runtime (detailed in Subsection 3.3.3).

From the client's viewpoint, Enarx's attestation protocols culminate in two cryptographically validated affirmations:

1. Verifying the specification and version of the TEE.
2. Confirming the integrity and version of the Enarx system.

It's crucial to underscore that the attestation procedures differ significantly across various hardware architectures such as in the SEV architecture and in SGX architecture.

3.3.2 Enarx API and Core:

The Enarx API and Core project establishes the WASI APIs and oversees attestation across various TEEs. This initiative presents a unified platform for developers to engage with Enarx Keeps, guaranteeing the integrity of the TEE and safeguarding against any tampering of the Enarx runtime. Executed in Rust, the Enarx API and Core assume responsibility for the upper-level functionalities of Enarx [45].

3.3.3 Enarx Runtime:

The Enarx runtime depends on WASI as a call-out API and Wasm as a JIT (Just-in-Time) compiler [45].

WASI acts as a standardized interface, facilitating smooth communication between WebAssembly modules and the host system. This ensures that the Enarx runtime can provide a consistent and secure environment for applications to function within diverse TEEs. WASI's comprehensive capabilities, covering crucial functionalities like file I/O, network sockets, and timers, prove highly advantageous for running applications within TEEs [143].

Furthermore, the Enarx runtime utilizes Wasm as its JIT compiler. JIT compilation, a method of converting code into machine code during runtime, is well-suited for Enarx runtime due to Wasm's portability across various TEEs. This approach enables the deployment of Enarx applications on a wide array of platforms. Since Wasm is a compiled language, it delivers more efficient execution compared to interpreted languages, a crucial factor for TEE-based applications where performance is paramount [**Wasmtime**].

3.3.4 Management:

The Management component in the Enarx framework plays a crucial role by orchestrating and managing workloads in Enarx Keeps [45]. It offers carefully designed APIs to simplify application deployment, supervision, and monitoring within Enarx's environment.

The Management component operates within a Kubernetes cluster as an amalgamation of microservices. It achieves seamless interaction with the Enarx API and Core through a well-defined gRPC interface [3]. Moreover, it employs the Enarx Attestation component to affirm the authenticity of devices executing Enarx Keeps, thereby enhancing overall security. Additionally, it engages in collaborative endeavours with the Enarx runtime component to proficiently oversee application execution within Enarx Keeps.

Notwithstanding its intricate architecture, the Management component places paramount

[3] https://grpc.io/

emphasis on user-friendliness. It provides APIs similar to those of Kubernetes, tailored to developers familiar with Kubernetes practises and facilitating rapid mastery of its features. Furthermore, the Management component meticulously aligns itself with the exigencies of enterprises and organizations, underscoring its strategic commitment to advancing Enarx adoption [45].

3.4 Working Method

When considering the developer's perspective, the process for deploying an application on Enarx includes selecting a language, developing the application, compiling the program into a Wasm binary, and finally deploying the application onto the designated Host. The uncomplicated deployment procedure of an application using Enarx is depicted in Figure 3.5.

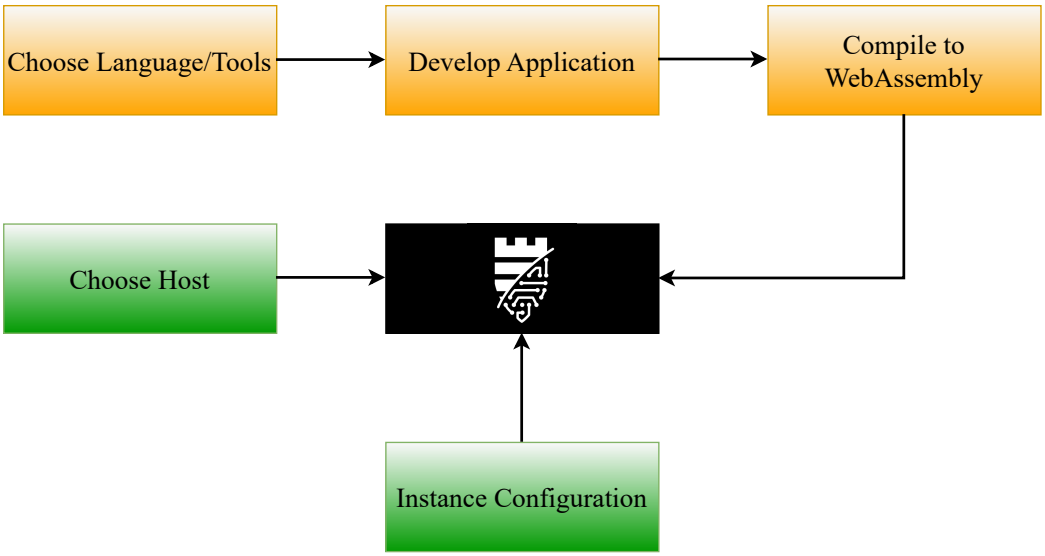

Figure 3.5. *Streamlined Flowchart Illustrating Application Deployment on Enarx*

Enarx facilitates the intricate procedures required to guarantee the secure deployment of applications within authentic TEE instances. Figure 3.6 presents an alternative perspective of Figure 3.4, elucidating the operational methodology of Enarx's attestation process.

Initiating from the initial phase, users or administrators commence a request for workload placement via the CLI of the Enarx Client Agent. The Enarx client agent engages with the host agent, a server that collaborates with the CPU and firmware to establish a foundational enclave. This is the attestation step, where Enarx verifies that the host is genuine and can be trusted to run the workload in a secure environment. The interaction between the client agent and the host agent takes place through a secure HTTPS connection,

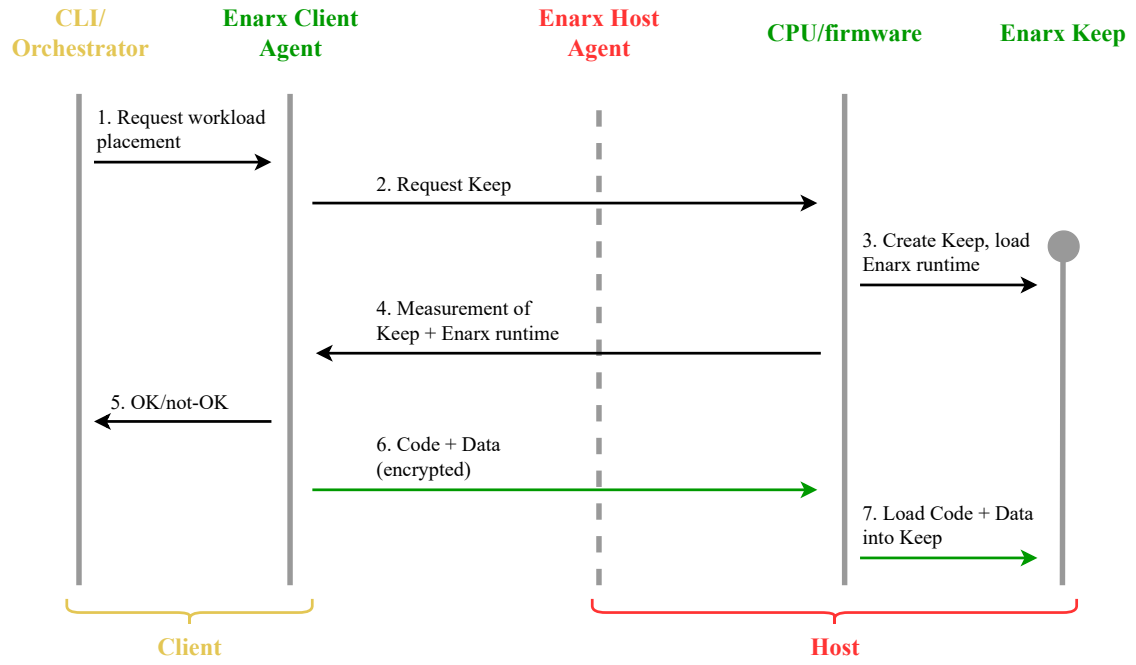

Figure 3.6. *Illustration of Enarx Attestation Process*

instilling users with the confidence that their requests are securely transmitted.

Once the Keep is created, the host agent sends a measurement of the Keep back to the client agent. The client agent checks this measurement to make sure that the Keep is genuine and has not been tampered with. Following successful verification, the Client Agent proceeds to encrypt the Wasm file using a session key exclusively tied to that specific enclave. This is the packaging step, where the workload is encrypted so that it cannot be tampered with by the host or by other software on the system.

With the encryption of the Wasm file concluded the client agent sends it to the Host Agent for execution within the keep. This is the provisioning step, where the workload is deployed to the keep and executed in a secure environment. It is essential to emphasise that every keep is associated with a unique session key. This indicates that the Wasm file encryption requires a unique key for each keep. Therefore, even the generation of another kept on the same system necessitates a completely distinct key for the Wasm file's encryption. This mechanism ensures the isolation of each keep, preventing any mutual visibility or data accessibility among concurrently running applications. With the encryption of the Wasm file concluded the client agent sends it to the Host Agent for execution within the keep.

These consecutive stages encompass the process through which Enarx establishes a secure computational environment

3.5 Vulnerabilities

Enarx relies upon TEE hardware to ensure confidentiality. The security of Enarx is closely intertwined with the vulnerabilities associated with TEE hardware such as Intel SGX or AMD SEV. The security posture of Enarx is adversely impacted by these vulnerabilities.

On the 16th of May, 2022, the Enarx community launched the Cryptle Hack Challenge, an initiative aimed at identifying vulnerabilities within the Enarx system [35]. During the course of this challenge, a flaw was uncovered in the implementation of Enarx.

To accommodate various CPU architectures, Enarx has employed distinct shims tailored for specific platforms. These shims incorporate small-scale μkernel, which in turn load executable components that are agnostic to CPU architectures. The communication between the Intel SGX shim and the host occurs through a designated block of host memory referred to as the sallyport block [4]. When the host accesses the SGX enclave, it provides a pointer to the sallyport block using the rdi register. Subsequently, when the shim necessitates executing syscalls or other Enarx-specific commands, it encodes the parameters into the sallyport block before returning control to the host.

Notably, on the 23rd of May, 2022, a vulnerability was reported within the Intel SGX shim [53]. The identified bug pertains to an absence of validation within the shim, concerning whether the pointer to the sallyport block, as furnished by the host, genuinely points to host memory rather than enclave memory. Exploiting this situation, the host can manipulate the shim into inadvertently corrupting its own memory by passing a pointer to the enclave's memory.

3.6 Practicalities

In the context of this thesis, while Enarx's documentation presents the deployment of diverse programs on varying hardware as a straightforward task, the practical implementation of this approach encounters significant challenges.

During our engagement with Enarx, we encountered unexpected obstacles, the first of which was the selection of hardware capable of fully harnessing TEE capabilities. Enarx is designed specifically for AMD SEV and Intel SGX, with Intel SGX II presenting specific hardware requirements. The crucial task of identifying suitable hardware for Enarx proved pivotal in utilizing the TEE features. Although Enarx offers development support for various hardware configurations, neglecting the security aspects of TEE would undermine the core objective of privacy and anonymity solution development.

Another critical consideration was the choice of programming language. While Enarx does support multiple programming languages, the primary criterion for language selec-

[4] https://docs.rs/sallyport/latest/sallyport/

tion revolved around the final compilation into Wasm binaries. Our attempts with languages such as C++ and Python revealed challenges in achieving a feasible compilation into Wasm binaries, particularly when working with various libraries. These challenges restricted our use of Enarx's preferred language, Rust.

Furthermore, due to Enarx's developmental phase, comprehensive documentation was lacking, necessitating direct communication with the development team to address any encountered errors.

4. DESIGNING PRIVACY EXTENSION FOR SEARCH ENGINES

This chapter delves into the intricacies of implementing a PESE to enhance privacy and anonymity in web search queries while mitigating the limitations. To begin with, we present an overview of the existing literature in this field, elucidating the challenges and opportunities associated with implementing privacy extensions in search engines. Following this, we outline the security considerations and the threat model that form the foundation of our design, along with our specific project goals and objectives. Subsequently, we delve into the project's design and methodology, providing an explanation of the technical features' implementation. Lastly, we delineate the limitations and biases encountered during our simulation.

4.1 Literature Review

Before developing the implementation of PESE, we conducted an evaluation of existing solutions for PESE. Our design is inspired by analogous works that have already been implemented. To identify relevant works, we conducted a thorough search of the scientific literature, focusing on peer-reviewed studies that were publicly accessible. We excluded any private industry solutions whose implementation details were not available to the public. Our search was conducted using the following search engines:

- *Andor*[1]
- *ACM Digital Library*[2]
- *arXiv open-access archive*[3]
- *dblp computer science bibliography*[4]
- *IEEE Xplore*[5]
- *Google Scholar*[6]

[1] https://andor.tuni.fi/
[2] https://dl.acm.org/
[3] https://arxiv.org/
[4] https://dblp.org/
[5] https://ieeexplore.ieee.org/
[6] https://scholar.google.com/

The acronym *PESE* is not typically associated with the privacy extension for search engines and is created by ourselves. In the Finnish language, *PESE* translates to *"Wash,"* signifying the act of eliminating traces. *PESE* serves as a representation of erasing digital footprints to protect users from online surveillance and data breaches, thereby strengthening the concept of privacy and anonymity.

To assure a thorough search, we employed the following search terms and their combinations: *"Privacy", "Extension", "for", "Search", "Engine", or "Privacy", "Web", "Search."* By employing these criteria, our goal was to obtain relevant literature and research related to privacy extensions for search engines that align with our design approach for PESE. It is crucial to acknowledge that numerous methods and efforts have been dedicated to protect user web privacy, such as CoFeed [47], OSLo [147], and homomorphic encryption methods [157, 73]. However, our focus is specifically on solutions and research endeavors that align with our interest and approach, and have served as motivation for our design.

We identified a total of six research works that were closely aligned with our research objectives. These works provide perceptive data and serve as fundamental references for our research Table 4.1.

Title of the paper (abbreviated)	Authors	Year	Reference
TrackMeNot: Enhancing the privacy of Web Search	Al-Rfou', Jannen, and Patwardhan	2012	[6]
DisPA: An Intelligent Agent for Private Web Search	Juarez and Torra	2015	[78]
PEAS: Private, Efficient and Accurate Web Search	Petit et al.	2015	[109]
PaOSLo: Profile Aware ObScure Logging	Ullah et al.	2022	[148]
X-search: Revisiting private web search using Intel SGX	Mokhtar et al.	2018	[95]
CYCLOSA: Decentralizing Private Web Search Through SGX-Based Browser Extensions	Pires et al.	2018	[112]

Table 4.1. *Relevant PESE developments*

The first relevant contribution in this domain is TrackMeNot (TMN) [6], a Firefox extension specifically developed to preserve privacy during web searches. TMN achieves this objective by generating decoy queries to obfuscate users' genuine search terms [97]. To effectively emulate users' search behavior, TMN incorporates several mechanisms. These include the utilization of dynamic query lists and real-time search awareness (RTSA). Additionally, TMN maintains live header maps, which consist of dynamically updated variables representing the headers and URLs of the most recent search conducted by the browser. The design also employs the Burst-mode queries technique, which triggers a batch of queries in close proximity to actual user searches. Lastly, TMN intercepts and restricts cookies transmitted from the browser to the search engine if they are associated with a user's search request or subsequent response.

DisPA [78], is a browser extension that acts as a proxy between the user and the search engine. It semantically dissociates queries in real time, thereby increasing user privacy and making it more difficult for search engines to track user activity. It works by first

identifying and classifying the user's search query. Once the query has been classified, DisPA then generates a set of semantically related queries that can be used to search for the same information without revealing the original query. These queries are then sent to the search engine, and the results are returned to the user.

PEAS [109], a protocol designed to ensure privacy in Web searches, is the second pertinent research endeavor. The protocol implements indistinguishability by using the logical OR operator to combine each user query with k counterfeit queries. The design's distinctive trait is the generation of forged queries. It is based on a group profile, which is an aggregate of prior queries from a community of users. The group profile is published in a privacy-preserving manner by the privacy proxy. Subsequently, these requests are transmitted to the search engine via a privacy-preserving proxy, which guarantees unlinkability.

PaOSLo [148] a web search privacy-preserving protocol that mitigates the digital traces a user leaves in Web searching. PaOSLo systematically groups users based on profile similarity and assigns them to dynamic groups. This grouping ensures that the user's search history is mixed with the search histories of other users in the group, making it difficult for search engines to identify the user's individual interests.

X-Search [95], an Intel-SGX based private Web search mechanism, represents the fifth relevant work. The X-Search proxy receives the queries and executes them on behalf of the user, prior to forwarding them to the search engine. Within a trusted SGX enclave, the proxy executes verified code. The queries are encrypted outside the enclave and can only be accessed in plain text from within. To obscure the original query, the proxy combines the original query with k-random prior queries using the logical OR operator. This combination makes it difficult for the search engine to differentiate the original query. The obfuscated scheme modifies the returned search results by incorporating the results of both the initial query and the aggregated past queries. The designed proxy filters the results appropriately to ensure that only relevant results are delivered to the user.

CYCLOSA [112], a pertinent work comparable to the X-Search design, prioritizes security by leveraging Intel SGX's trusted execution environments. It generates fabricated queries to the search engine and modifies their frequency dynamically based on the sensitivity of the user's query. CYCLOSA operates in a decentralized fashion, delegating the task of transmitting fake queries to multiple nodes. To ensure unlinkability, each node functions act as both a client and as a proxy, initiating and forwarding queries on behalf of other nodes. Since these nodes are controlled by distinct users, they are deemed untrusted, effectively preventing any query information leakage. The design implements secured connections to increase the security of inter-enclave communications and interactions with the search engine.

Evaluating the relevant scholastic literature (also summarized in Table 4.2) reveals an inadequate amount of research on the privacy of web users with TEE. The majority of

Mechanism	Unlinkability	Obfuscation	Accuracy	TEE	
				Intel SGX	AMD SEV
TMN [6]	✗	✓	✓	✗	✗
DisPA [78]	✓	✓	✗	✗	✗
PEAS [109]	✓	✓	✗	✗	✗
PaOSLo [148]	✓	✓	✗	✗	✗
X-Search [95]	✓	✓	✗	✓	✗
CYCLOSA [112]	✓	✓	✓	✓	✗
PESE [79]	✓	✓	✓	✓	✓

Table 4.2. *Private web search mechanisms comparison*

TEE-based research focuses on specific hardware, such as Intel SGX. Nevertheless, PET (Privacy Enhancing Technologies) has emerged as an active research area, yielding numerous search results on topics including "web privacy," "query obfuscation," "anonymity," "digital traces," "privacy-preserving," and "privacy protection." This raises the subject of whether it is adding value to develop privacy-focused solutions using TEE or whether existing solutions suffice. User search queries often contain personally identifiable information and the accumulation and use of search queries reveal confidential information about individuals, such as their age, gender, religious or political preferences, and sexual orientation [133]. Numerous users advocate for search engines to refrain from retaining query logs [75]. EU data protection laws emphasize the right to privacy *(Article 7 of the Charter)* and the protection of personal data *(Article 8 of the Charter)* [40]. Consequently, addressing privacy concerns in web searches is crucial for users from diverse backgrounds, and search engines must comply with these regulations to avoid legal repercussions. In our view, PESE implementations utilizing TEE has substantial real-world value in protecting the privacy and anonymity of web users, necessitating further academic research.

4.2 System and Adversary Model

Before introducing the design principles of PESE in the upcoming section, we will outline our assumptions and the adversary model against which our protocol has been developed.

PESE performs computations based on two premises for each user query: (i) The Search Query Mixer, and (ii) the search engine. Different levels of trust are ascribed to each of these elements.

We presume that the user's device, which initiates search requests, is secure and trustworthy. This includes any local computations performed outside of Keeper as well as any

duties involving user data, such as determining the sensitivity of a query. In addition, we presume that the communication between the web user, query aggregator, and search engine, is secure and trustworthy. This means that the adversary cannot alter the inquiry input by the human user, nor communication channels be compromised.

The second assumption is that the hosting server supports Intel SGX II or AMD SEV. TEEs are utilized to secure the implementation from adversarial server root access. In addition, we presume that SGX or SEV behave correctly, implying that there are no flaws or backdoors present. We do not consider side-channel attacks [31] against Intel SGX [154] or AMD SEV [82] to be within the scope of our research because the research community offers solutions to counter such attacks. Furthermore, we have faith in the system's cryptographic primitives and libraries, presuming they cannot be tampered with.

Enarx's keeps execute all the code and data associated with PESE. This ensures that even with unrestricted access to the server, adversaries or administrators cannot access or manipulate the data, thereby enhancing the integrity and confidentiality of the system. Web users are not dependent on the server operator's trustworthiness or benevolence. They rely instead on the effective operation of TEE technology to prevent data observation or modification on the server. During our testing, we have full command over both the server and the web clients.

Thirdly, we presume that the search engine is both trustworthy and inquisitive. This indicates that the search engine faithfully responds to queries generated by the Search Query Mixer while attempting to assemble information from incoming queries. PESE was implemented and tested in this particular system and adversary model.

4.3 Objectives

The goals of our simulation can be categorized into three main objectives:

1. To establish a fully functional PESE system that incorporates elements to improve the privacy and confidentiality of web users.
2. To assess the practicality of PESE as a viable solution to address privacy and anonymization challenges.
3. To ascertain the feasibility of utilizing TEEs to enhance the security of the PESE system in real-world scenarios.

Consequently, our simulation aims to provide answers to specific research inquiries, namely RQ3 and RQ4. In Chapter 5 of our study, we will evaluate the extent to which our simulation achieves these objectives.

4.4 Design Approach and Methodology

PESE is an extension designed to enhance the level of anonymity and privacy of web search queries. Its main elements are the Keeper, the Search Engine, and web users. PESE aims to improve privacy and anonymity by combining the concepts of unlinkability and indistinguishability. Unlinkability and Indistinguishability are accomplished by mixing the queries from web users within the Keeper, making them virtually indistinguishable from one another. From the user's perspective, the interaction with the search engine remains unchanged, while the rest of the operations are performed within a black box. However, from the server's perspective, the architecture becomes more complicated as it must communicate with web users requesting queries, conduct operations within the Keeper for obfuscation purposes, and return search results to web users.

PESE employs a multi-step approach. When web users submit a query to the search engine, they send an HTTP GET request. In our experiment, we utilized *Ahmia.fi* [7] as the search engine. The number of users can vary, and let's denote the total number of users as n. The queries sent by these users are represented as $q_1, q_2, q_3, ..., q_n$, where Q denotes the total number of queries. The communication between the user, Keeper, and search engine is established via a secure connection TLS. These queries are then forwarded to the server, where operations are performed within the Keeper.

Any resources outside of the Keeper cannot be trusted upon. Within the Keeper, the integrity of the search engine is first verified through the use of checksums. The Keeper decouples users' IP addresses from their search phrases, ensuring search activities are not directly linked. It receives incoming queries over a specified duration of time, denoted as t, and creates a thread to handle these requests. The degree of anonymity is determined by the number of users making search queries. The queries are aggregated in a random order fashion within the Keeper. By aggregating queries in an arbitrary sequence, the Keeper precludes the identification of individual users based on the order of their search queries.

After the time delay Δt, the query pulses are directed to the search endpoint, and the search engine results $(r_1, r_2, r_3, ..., r_n)$ corresponding to the queries $(q_1, q_2, q_3, ..., q_n)$ are returned to the Keeper. Within the Keeper, an identification hash of the search result is generated using the *SHA512* [66] cryptographic hash. This approach ensures that result pages contain padding to prevent naive size correlation, thereby ensuring that all result pages have the same byte size. Finally, the web browser makes another HTTP GET request to fetch the results, maintaining no connection between the incoming and outgoing traffic during these calls.

Figure 4.1 illustrates the design of PESE, describing the systematic implementation of its

[7] https://ahmia.fi/

components, Keeper, the Search Engine, and web users. Furthermore, the PESE source code is included as an appendix for developers' reference in order to better comprehend the technical aspects.

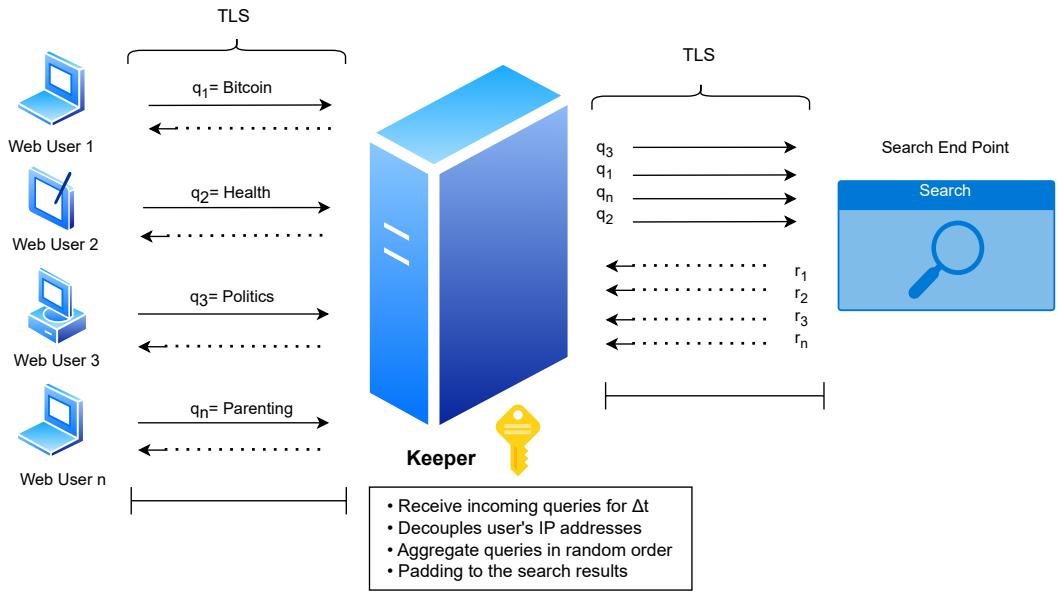

Figure 4.1. *Design of Privacy Extension for Search Engines*

PESE's design is meticulously crafted to ensure a higher level of anonymity. By aggregating search queries within the Keeper and executing them in a randomized sequence, PESE effectively obscures the identification of individual users. To illustrate, if we consider a scenario where the solution utilizes the Google search engine, handling over 3.5 billion queries daily [95]. Even with a minimal time delay, such as 100 milliseconds, PESE can create a substantial anonymity set comprising more than 1,000 individuals seamlessly mixed together. Consequently, the search engine becomes incapable of associating specific users with particular search terms, as the search requests are intentionally delayed and blended into a collective stream. The assumption that 1,000 people are concurrently doing searches using various queries is derived from statistical data collected from multiple search engines [89].

We conducted simulations in an environment using an AMD SEV server, which acts as a proxy between web users and the search engine. To ensure utmost security, we employed Enarx to execute the simulation within a TEE. The utilization of a TEE guarantees the integrity and confidentiality of processed data, securing it against any compromise of the underlying server. Even with root access, the high-level operating system remains unable to access or modify the data, ensuring robust protection.

4.5 Limitations and Potential Bias

While PESE prioritizes user privacy however sacrifices personalized search, as its obfuscation process prevents tailored results from user profiles, interests, or locations. Furthermore, the effectiveness of the search query mixer within PESE relies on the presence of other users' real queries received within the Keeper during a specific timeframe. In the event that no other user submits a query, a single query alone is forwarded to the endpoint, rendering it susceptible to correlation attacks [137]. Additionally, if one of the search queries happens to be in a different language, it introduces the potential for re-identification attacks [62].

To address these issues, the introduction of fake queries in the PESE system would be a beneficial feature. Furthermore, these fake queries should be generated in various languages to enhance the diversity of the query set and mitigate the risk of re-identification attacks. Furthermore, to counter the potential threat of a malicious process reproducing previous user queries, a precautionary measure involves integrating a random identifier into each message. This implementation serves the purpose of detecting and preventing replay attacks, enhancing the overall security of the system.

During the course of our simulation, we make an assumption that the server is trustworthy and the operations are executing securely within the Keeper. However, Enarx does not yet have a mechanism that allows users to validate the attestation of the TEE, which is essential for assuring that users are communicating with genuine TEE instances. In the event of a compromised TEE, users may unknowingly continue sending queries under the assumption of a legitimate TEE environment, while administrators or adversaries gain insight into these processes.

It is important to note that while TEEs provide significant advantages in terms of data confidentiality, integrity, and authenticity, they may still be susceptible to SCAs. Manufacturers often exclude SCAs from their TEE threat models [76], and TEEs may not adequately protect against SCAs in real-world scenarios. Protecting against specific SCAs requires extensive research to address vulnerabilities. Nevertheless, ensuring comprehensive safeguarding against all conceivable SCAs proves unattainable, TEEs do not genuinely deliver the level of design-based security assurances they seemingly promise [159, 128]. Despite this, the use of TEEs helps reduce the system's attack surface, acting as a deterrent against most attackers except highly skilled and resource-rich adversaries. Although SCAs pose a theoretical risk, their real-world exploitation against TEE-enabled systems is uncommon, making the use of TEEs a valuable strategy to enhance system security.

To achieve a genuinely secure implementation, it is essential to recognise and address the biases and limitations outlined above. Maximising the effectiveness of TEEs and ensur-

ing robust system security requires ongoing research, software modifications, and careful consideration of potential vulnerabilities. For a truly secure implementation, a comprehensive approach that addresses each of these biases and limitations is essential.

5. RESULTS OF PESE DEVELOPMENT AND TESTING

This section provides a description of the methods used to test our PESE implementation, the degree of anonymity as well and an overview of the issues and considerations encountered throughout the testing process.

5.1 Performance Evaluation

We evaluated the PESE implementation by deploying it on AMD server hardware equipped with 128-core CPUs and 250 GB of RAM. In this scenario, the search engine of choice was *Ahmia.fi* [1]. The testing procedure involved executing search queries comprising 50 words, simulating requests from multiple users generated from the same system. This process was iterated 50 times in a loop to monitor its impact on the server and to detect any anomalies in the outcome of the results. This procedure was carried out under three distinct conditions:

1. Enarx with SEV as background utilizing TEE
2. Enarx with NIL background not utilizing TEE
3. Cargo Environment without utilizing TEE

Furthermore, within the same environment, we conducted stress testing to ascertain the system's stability when subjected to increased workload.

The performance metrics, as outlined in Table 5.1, encompassed measurements of CPU cycles, elapsed time, user time, system time, and RAM consumption for each individual test. Each test query encompassed a set of 50 unique terms, and the entire testing process was repeated 50 times, with results recorded on each occasion. In order to assess the effectiveness of the privacy extension, we derived the mean values from the collected results.

Furthermore, we plot the graph bar and line plot as shown in Figure 5.1 and Figure 5.2, against the observed values of performance metrics for each environment in order to make easy visual comparison.

In the context where Enarx is employed with SEV as the backend, the script exhibits an ex-

[1] https://ahmia.fi/

Environment/ Performance Metrics	CPU Cycles	Time Elapsed (sec)	User Time (sec)	System Time (sec)	RAM (MB)
SEV BG	677215821	53.36827648	0.29410246	0.07721462	3.93
NIL BG	677713078	51.19152534	0.32286962	0.09149396	3.96
Cargo	690029232	56.11155829	0.29282962	0.07125856	3.91
Stress SEV BG	968461379	54.74515538	0.39104184	0.08693166	3.92
Stress NIL BG	949583166	53.83668458	0.36363614	0.10100244	3.95
Stress Cargo	955451884	63.28933305	0.38091528	0.0843392	3.91

Table 5.1. *PESE Environment Performance Metrics*

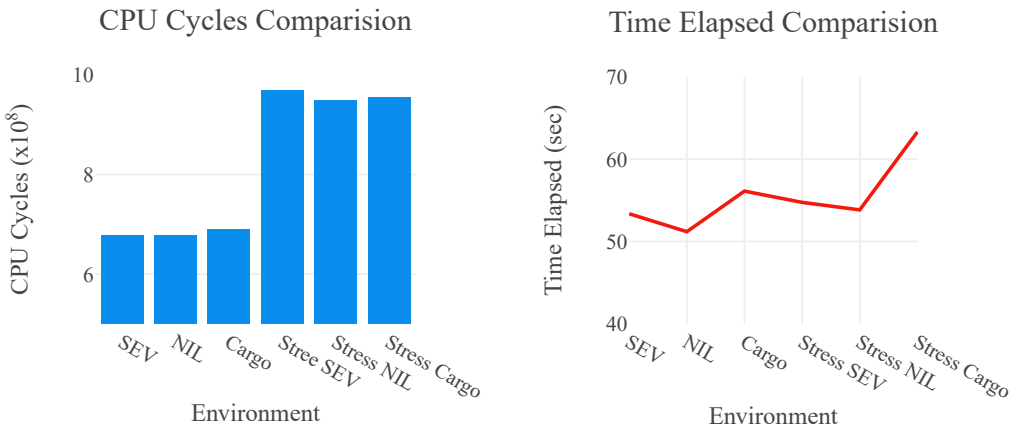

Figure 5.1. *CPU Cycles and Time Elapsed plot for each testing environment*

tended execution duration, encompassing *677,215,821* CPU cycles that span over *53.37* seconds. This elongated execution timeframe can be attributed to the incorporation of enhanced security measures afforded by the TEE. In direct contrast, the scenario devoid of TEE technology with NIL as the backend showcases a marginal enhancement in performance, culminating in a completion time of *51.19* seconds while consuming *677,713,078* CPU cycles. In this context, the absence of TEE-related overhead contributes to the observed acceleration; however, the trade-off lies in compromised security. It is essential to acknowledge that the elapsed time refers to the whole period of the script's execution, including both the process of making queries and the subsequent observation of results. It does not indicate the response time of the corresponding search result.

A comparative analysis is also undertaken against a conventional Rust Cargo environment running Wasm binary, which serves as a benchmark. Within this benchmark, the script reaches its conclusion in *56.11* seconds, propelled by *690,029,232* CPU cycles. This comparative evaluation serves to contextualize the discernible influence of Enarx and TEE on overall performance.

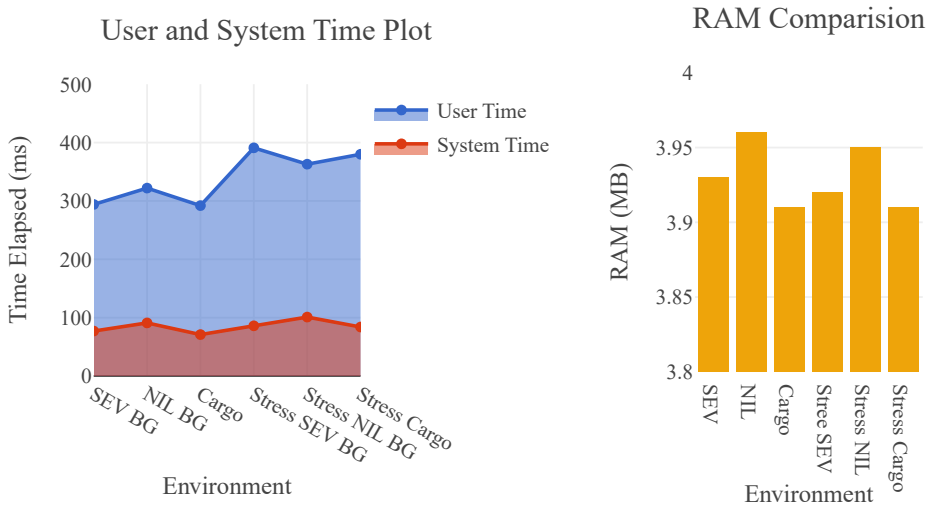

Figure 5.2. *User Time, System Time and RAM consumption plot for each testing environment*

The application of stress testing yields further insights. In scenarios where Enarx is engaged in tandem with SEV backend, the script's execution time extends to *54.75* seconds under stress, accompanied by a consumption of *968,461,379* CPU cycles. Conversely, stress tests conducted without TEE through the NIL backend yield a completion time of *53.84* seconds, entailing *949,583,166* CPU cycles.

An intriguing observation arose from the significantly extended elapsed time recorded within the Rust Cargo environment running Wasm binary. This outcome diverged from our initial anticipation of lower values, prompting a notable discrepancy. This behaviour can be rationalized due to the runtime environment, as substantiated by our prior investigation detailed in our Sok-TEE research paper attached in Appendix. However, on the whole, the findings deduced from the outcomes signify that Enarx with SEV slightly influences the performance of the system running PESE. Nevertheless, the observable effect on performance does not reach a level of importance that undermines the essential security and privacy benefits provided by Enarx.

5.1.1 Degree of Anonymity

The degree of anonymity serves as a metric gauging the complexity of identifying a user or actor within a system or network. PESE's design is meticulously tailored to elevate the overall level of anonymity. As depicted in Figure 4.1, the search query mixer aggregates search queries and executes them in a randomized sequence, effectively concealing the identification of individual users. The degree of anonymity factors in the likelihood associated with each user's query activity during specific time intervals.

The testing methodology employed in our research comprised the execution of search

queries, each consisting of 50 words. These queries were simulated to emulate requests from multiple users originating from the same system. The target of these search queries was Ahmia.fi search engine. However, in the context of the substantial search query volumes handled by popular platforms such as Google, which manages over 3.5 billion queries daily [95] and processes 3.8 million search queries per minute, alongside Yahoo processing 63.65 thousand search queries per minute and Bing processing 121 thousand search queries per minute [89], PESE's significance becomes apparent.

Even with a brief temporal lag of 100 milliseconds, a considerable level of confidentiality can be attained by seamlessly combining over 1,000 individuals. Consequently, the search engine's ability to correlate specific users with particular search terms is compromised, given that PESE intentionally introduces delays and blends queries into a collective stream.

PESE demonstrates independence from the generation of spurious queries from a predefined list, avoiding the maintenance of user profiles or scrutiny of similarities among diverse user search queries that might inadvertently result in the creation of shadow profiles during processing. This distinctive approach provides PESE with a significant advantage over alternative solutions shown in Table 4.1. Even solutions relying on TEE for data protection during processing are often limited to specific hardware like Intel SGX. PESE leverages the Enarx framework. This adaptability facilitates seamless redeployment on AMD SEV as well. A comprehensive comparison of PESE with other solutions, highlighting its advantageous features, is detailed in Table 4.2 as well.

PESE employs a multifaceted approach to thwart re-identification attacks and enhance user privacy. The system aggregates and randomizes search queries within the Keeper, introducing deliberate randomness that makes it challenging for external entities, including search engines, to associate specific users with their search terms. A time delay (Δt) further disrupts the correlation between incoming queries and outgoing search results, creating a substantial anonymity set. Result pages undergo cryptographic hashing, incorporating padding to prevent size-based correlation. Additionally, the Keeper decouples users' IP addresses from their search phrases, hindering direct links between user identities and search activities. Simulations within a TEE ensure data integrity and confidentiality, guarding against server compromises. Together, these measures form a strong defence mechanism, rendering re-identification attacks difficult and reinforcing PESE's commitment to user anonymity and privacy.

5.2 Practical Consideration and Potential Issues

Enarx and PESE both currently exist in their nascent stages. However, upon introduction to the testing environment, several challenges were encountered. As elucidated in Section 3.1, Enarx strategically keep networking outside the TCB to enhance the security

architecture. Nevertheless, this approach has presented difficulties during the development of networking-centric applications. During the implementation of a fundamental TCP client using sockets, for instance, we encountered numerous complications.

The core of Enarx's efficacy lies in its data-plane isolation feature, mandating that applications be purposefully crafted to communicate through Enarx's secure channels. This requirement can prove to be intricate for applications not initially designed with Enarx integration in mind. While Enarx offers advantages such as versatile hardware deployment and multi-language support, it is important that Rust constitutes its primary programming language. An advantageous aspect of this is that Rust code can be conveniently compiled into Wasm binaries, which is the execution environment for Enarx. However, for languages other than Rust, the process of compiling to Wasm binaries is difficult, prompting developers to gravitate towards Rust over time. Furthermore, the absence of comprehensive and up-to-date documentation for Enarx creates challenges in troubleshooting and often necessitates direct engagement with Enarx's development team for issue resolution.

6. CONCLUSION

In this thesis, the exploration commenced with an extensive examination of TEE and diverse solutions addressing privacy and anonymity. Subsequently, a thorough investigation into the Enarx framework unfolded, encompassing intricate details of its implementation and technical specifications. A notable contribution to this thesis lies in the development of the PESE using the Enarx framework. The primary objective of this development was to elevate user privacy and anonymity by concealing online web queries within a TEE and overarching goal throughout the thesis remained consistent.

In this concluding chapter, comprehensive responses are provided to the research questions outlined in Section 1.3. Furthermore, a discussion on potential future research avenues unfolds, presenting open questions that emanate from the insights gained during the development and scrutiny of PESE within the Enarx framework. This collective exploration contributes to the broader objective of advancing the realm of privacy and anonymity solutions.

6.1 Answers to Research Questions

We presented the following research questions in Section 1.3

RQ1. What is the importance of TEE in augmenting the security of data and operational processes?

RQ2. How do conventional solutions address privacy and anonymity concerns for web users?

RQ3. How effective is the Enarx framework for developing privacy-enhanced application PESE?

RQ4. What are the results and conclusions regarding the impact of PESE on the privacy and anonymity of web users?

For RQ1, a comprehensive analysis in Chapter 2 unfolds from Section 2.1 to Section 2.4, shedding light on the paramount importance of TEEs in fortifying the security of sensitive data and operational processes. The examination begins in Section 2.1, elucidating the pivotal role of TEEs in mitigating potential risks. TEEs, by design, establish isolated and secure enclaves, serving as strong fortresses protecting critical data and processes

against unauthorized access and manipulation. This intrinsic capability positions TEEs as formidable defenses, even in scenarios where adversaries gain root access to the system. Furthermore, the discourse in Section 2.2 provides nuanced insights into diverse application categories that stand to benefit significantly from the integration of TEEs. Delving deeper, Section 2.3 explores the fundamental components underpinning the secure functionality of TEEs. Emphasis is placed on the necessity of tailoring solutions to align with the unique architectural features of the underlying hardware. Addressing the challenges associated with hardware-agnostic TA deployment, Section 2.4 presents an examination of TCons up to the year 2022. This examination underscores the limitations linked to enabling seamless TA deployment across varied hardware architectures without substantial modifications.

When considering RQ2, Section 2.5 to Section 2.8 in Chapter 2 provides a thorough examination of privacy and anonymity concerns for web users, delving into conventional solutions and their efficacy. The central challenges revolve around the vast amount of personal data generated particularly through WSEs and the potential risks associated it. Conventional solutions predominantly adopt two key strategies: unlinkability and indistinguishability. Unlinkability, exemplified by anonymous communication protocols like Tor and I2P, which aims to cloak user identities. The indistinguishability approach, represented by solutions like TrackMeNot and GooPIR, seeks to obfuscate the distinction between authentic and fabricated queries. The historical exploration within the chapter underlines the evolution of privacy and anonymity concepts, emphasizing their roots in ancient societies and their incorporation into modern legal frameworks. The chapter further investigates the evolution of privacy and anonymity systems, beginning with remailing systems in the 1990s and progressing to contemporary technologies such as Tor and I2P. The development of Bitcoin mixers in response to privacy concerns in cryptocurrency transactions showcases the ongoing efforts to adapt privacy solutions to the dynamic digital landscape. In essence, the findings emphasize the intricacies and challenges associated with preserving web user privacy and anonymity. Conventional solutions, while contributing valuable insights and methods, face persistent hurdles. The chapter's exploration underscores the need for ongoing research and innovation to navigate the evolving nature of digital interactions and effectively address privacy concerns in the contemporary online environment.

In addressing RQ3, Chapter 3 undertakes a thorough examination of the Enarx framework, encompassing its design principles, requirements, core components, operational approach, and vulnerabilities. The study delves into how Enarx facilitates the advancement of TAs through the integration of TEEs. Enarx architectural foundation rests on the principles of trust, security, and isolation, adopting a layered approach to provide unique sets of security features based on a minimal trusting base. The scope of Enarx's operations spans attestation, secure boot, and runtime isolation processes. The primary

achievement of this thesis lies in a heightened understanding of the fundamental principles of the Enarx architecture and its pivotal role in TA development using TEEs, as evidenced in the selection of Enarx for the PESE project (discussed in Chapter 4). However, challenges arose during the development of PESE, revealing practical difficulties not initially apparent in Enarx's documentation. Critical tasks, such as identifying suitable hardware and choosing a programming language for compilation into Wasm binaries, and lack of documentation demanded careful consideration. The realization emerged that, despite its apparent ease of use, Enarx's current early development stage poses underlying challenges, a crucial insight for developers aiming to utilize Enarx for secure application creation.

In the context of addressing RQ4, Chapter 4 meticulously explores the implementation of PESE, providing an in-depth examination of the system model and design methodology. The foundation of PESE's design decisions is rooted in a thorough analysis of existing solutions and their limitations, as elucidated in Section 4.1. Notably, this analysis scrutinizes aspects of unlinkability, obfuscation, accuracy, and TEE hardware support. Each of the existing solutions under review is found to lack one or more crucial features, thereby impacting user privacy, anonymity, or the feasibility of deployment. Moving forward, Chapter 5 presents the outcomes derived from simulations conducted within a controlled operational environment, showcasing PESE's performance under representative conditions. It is crucial to note that the testing environment deliberately omits considerations such as SCA and user verification of TEE authenticity. PESE employs a comprehensive strategy for improving user privacy by leveraging the Keeper to aggregate and randomize search queries, introducing intentional randomness to thwart association between users and their search terms. A time delay parameter enhances anonymity by disrupting correlations between incoming queries and outgoing search results. To enhance security, result pages undergo cryptographic hashing with padding, preventing size-based correlation. The Keeper further severs the direct link between users' IP addresses and search phrases. Simulations within a TEE ensure data integrity and confidentiality, forming a strong defense mechanism that significantly enhances user privacy and anonymity through the PESE. While the test outcomes align with our expectations, there remains room for enhancements to facilitate PESE's deployment in real-world scenarios.

6.2 Future Research and Open Questions

Based on the limitations outlined in Section 4.5 and the conclusions drawn in the preceding Section 6.1, further research and development are imperative for PESE to enhance its functionality and security.

- In our existing system, a specific time is designated for accepting user queries. However, this can be refined by dynamically adjusting the time based on the number

of queries received. This entails that if a certain threshold of queries is reached before the predetermined time elapses, the query mixer will proceed without delay, thus augmenting system performance.

- As stipulated in the previous subsection, the absence of a mechanism for users to authenticate the TEE instance remains a significant concern in the context of TEE utilization. The absence of authentic TEE verification renders the system ineffective in safeguarding the privacy and confidentiality of web users.
- The protection of PESE against SCAs is another critical aspect. Notably, CPU manufacturers often exclude SCAs from their TEE threat models, and numerous real-world TEE vulnerabilities corroborate this vulnerability. Given our lack of supplementary SCA defenses, it is prudent to acknowledge the susceptibility of our system to SCA attacks.
- Our current system relies on Rust synchronous operations, but transitioning to asynchronous operations will yield performance improvements for PESE while concurrently mitigating system overhead.

Nonetheless, our PESE implementation introduces a distinctive approach to enhance user privacy and anonymity through TEE utilization, a method that can be further refined by diligently adhering to the aforementioned steps.

REFERENCES

[1] *About the Confidential Computing Consortium*. Confidential Computing Consortium. URL: https://confidentialcomputing.io/about/.

[2] Marshall D. Abrams and Albert B. Jeng. “Network security: Protocol reference model and the trusted computer system evaluation criteria”. In: *IEEE Netw.* 1.2 (1987), pp. 24–33. DOI: 10.1109/MNET.1987.6434188. URL: https://doi.org/10.1109/MNET.1987.6434188.

[3] *Act on the Protection of Privacy in Electronic Communications*. https://www.finlex.fi/en/laki/kaannokset/2004/en20040516_20110365.pdf. 2004.

[4] Administrative Office of the U.S. Courts. *What Does the Fourth Amendment Mean?* 1998. URL: https://www.uscourts.gov/about-federal-courts/educational-resources/about-educational-outreach/activity-resources/ (visited on 02/26/2023).

[5] AevaOnline and Squillace. *Mystikos*. Latest release in 2021. 2021. URL: https://github.com/squillace/mystikos (visited on 02/13/2023).

[6] Rami Al-Rfou’, William Jannen, and Nikhil Patwardhan. “TrackMeNot-so-good-after-all”. In: *CoRR* abs/1211.0320 (2012). arXiv: 1211.0320. URL: http://arxiv.org/abs/1211.0320.

[7] Afzaal Ali et al. “TOR vs I2P: A comparative study”. In: *IEEE International Conference on Industrial Technology, ICIT 2016, Taipei, Taiwan, March 14-17, 2016*. IEEE, 2016, pp. 1748–1751. DOI: 10.1109/ICIT.2016.7475027. URL: https://doi.org/10.1109/ICIT.2016.7475027.

[8] *AMD Secure Encrypted Virtualization (SEV)*. https://www.amd.com/en/developer/sev.html. Accessed in 2023. AMD.

[9] *An introductory overview of the Intel TDX technology*. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. White paper, Accessed in 2023. Intel, Feb. 2023.

[10] Nicolas Anciaux et al. “Trusted Cells: A Sea Change for Personal Data Services”. In: *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013. URL: https://api.semanticscholar.org/CorpusID:18273458.

[11] Apache. *Incubator Teaclave*. github repo. Latest release in 2023, URL accessed on 2023-07-28. 2019. URL: https://github.com/apache/incubator-teaclave.

[12] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. “A Secure and Reliable Bootstrap Architecture”. In: *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*. IEEE Computer Society, 1997, pp. 65–71. DOI:

10.1109/SECPRI.1997.601317. URL: https://doi.org/10.1109/SECPRI.1997.601317.

[13] Ghada Arfaoui, Said Gharout, and Jacques Traoré. “Trusted Execution Environments: A Look under the Hood”. In: *2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2014, Oxford, United Kingdom, April 8-11, 2014*. IEEE Computer Society, 2014, pp. 259–266. DOI: 10.1109/MobileCloud.2014.47. URL: https://doi.org/10.1109/MobileCloud.2014.47.

[14] Sergei Arnautov et al. “SCONE: Secure Linux Containers with Intel SGX”. In: *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. Ed. by Kimberly Keeton and Timothy Roscoe. USENIX Association, 2016, pp. 689–703. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

[15] Will Arthur, David Challener, and Kenneth Goldman. “History of the TPM”. In: *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Berkeley, CA: Apress, 2015, pp. 1–5. ISBN: 978-1-4302-6584-9. DOI: 10.1007/978-1-4302-6584-9_1. URL: https://doi.org/10.1007/978-1-4302-6584-9_1.

[16] N. Asokan. “Hardware-assisted Trusted Execution Environments: Look Back, Look Ahead”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro et al. ACM, 2019, p. 1687. DOI: 10.1145/3319535.3364969. URL: https://doi.org/10.1145/3319535.3364969.

[17] N. Asokan et al. “Mobile Trusted Computing”. In: *Proc. IEEE* 102.8 (2014), pp. 1189–1206. DOI: 10.1109/JPROC.2014.2332007. URL: https://doi.org/10.1109/JPROC.2014.2332007.

[18] Ahmed M. Azab et al. “Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM, 2014, pp. 90–102. DOI: 10.1145/2660267.2660350. URL: https://doi.org/10.1145/2660267.2660350.

[19] Ranjbar A. Balisane and Andrew C. Martin. “Trusted execution environment-based authentication gauge *(TEEBAG)*”. In: *Proceedings of the 2016 New Security Paradigms Workshop, NSPW 2016, Granby, Colorado, USA, September 26-29, 2016*. Ed. by Carrie Gates et al. ACM, 2016, pp. 61–67. DOI: 10.1145/3011883.3011892. URL: https://doi.org/10.1145/3011883.3011892.

[20] Kevin S. Bauer. “Improving Security and Performance in Low Latency Anonymous Networks”. PhD thesis. University of Colorado at Boulder, 2011.

[21] Jethro Beekman. *Fortanix, Runtime Encryption Technology*. Fortanix, 2020. URL: https://www.intel.com/content/dam/www/public/us/en/documents/research/jethro-beekman-edp-2020-update-fortanix.pdf (visited on 2023).

[22] Song Bian, Weiwen Jiang, and Takashi Sato. "Privacy-Preserving Medical Image Segmentation via Hybrid Trusted Execution Environment". In: *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*. IEEE, 2021, pp. 1347–1350. DOI: 10.1109/DAC18074.2021.9586198. URL: https://doi.org/10.1109/DAC18074.2021.9586198.

[23] Mike Bursell. *Trust No One, Run Everywhere–Introducing Enarx*. 2019. URL: https://next.redhat.com/2019/08/16/trust-no-one-run-everywhere-introducing-enarx/ (visited on 07/16/2023).

[24] Mike Bursell and Nathaniel McCallum. *Enarx intro for SGX workshop*. https://www.intel.com/content/dam/www/public/us/en/documents/research/mike-bursell-enarx-intro-for-sgx-workshop.pdf. Accessed in 2023.

[25] Marcel Busch, Johannes Westphal, and Tilo Müller. "Unearthing the TrustedCore: A Critical Review on Huawei's Trusted Execution Environment". In: *14th USENIX Workshop on Offensive Technologies, WOOT 2020, August 11, 2020*. Ed. by Yuval Yarom and Sarah Zennou. USENIX Association, 2020. URL: https://www.usenix.org/conference/woot20/presentation/busch.

[26] Emin Çalışkan, Tomáš Minárik, and Anna-Maria Osula. *Technical and legal overview of the Tor anonymity network*. NATO Cooperative Cyber Defence Centre of Excellence. 2015. URL: https://ccdcoe.org/uploads/2018/10/TOR_Anonymity_Network.pdf (visited on 2023).

[27] Claude Carlet and Sylvain Guilley. "Statistical properties of side-channel and fault injection attacks using coding theory". In: *Cryptogr. Commun.* 10.5 (2018), pp. 909–933. DOI: 10.1007/s12095-017-0271-4. URL: https://doi.org/10.1007/s12095-017-0271-4.

[28] Claude Castelluccia, Emiliano De Cristofaro, and Daniele Perito. "Private Information Disclosure from Web Searches". In: *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*. Ed. by Mikhail J. Atallah and Nicholas J. Hopper. Vol. 6205. Lecture Notes in Computer Science. Springer, 2010, pp. 38–55. DOI: 10.1007/978-3-642-14527-8_3. URL: https://doi.org/10.1007/978-3-642-14527-8_3.

[29] Shuo Chen, Jun Xu, and Emre Can Sezer. "Non-Control-Data Attacks Are Realistic Threats". In: *Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, July 31 - August 5, 2005*. Ed. by Patrick D. McDaniel. USENIX Association, 2005. URL: https://www.usenix.org/conference/14th-usenix-security-symposium/non-control-data-attacks-are-realistic-threats.

[30] Shuyi Chen, Yingyou Wen, and Hong Zhao. "Formal Analysis of Secure Bootstrap in Trusted Computing". In: *Autonomic and Trusted Computing, 4th International*

Conference, ATC 2007, Hong Kong, China, July 11-13, 2007, Proceedings. Ed. by Bin Xiao et al. Vol. 4610. Lecture Notes in Computer Science. Springer, 2007, pp. 352–360. DOI: 10.1007/978-3-540-73547-2_37. URL: https://doi.org/10.1007/978-3-540-73547-2_37.

[31] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. “Real time detection of cache-based side-channel attacks using hardware performance counters”. In: *Appl. Soft Comput.* 49 (2016), pp. 1162–1174. DOI: 10.1016/j.asoc.2016.09.014. URL: https://doi.org/10.1016/j.asoc.2016.09.014.

[32] CNBC. *What Is Anonymous? The Group Went from 4chan to Cyberattacks on Russia*. 2022-03-25. URL: https://www.cnbc.com/2022/03/25/what-is-anonymous-the-group-went-from-4chan-to-cyberattacks-on-russia.html (visited on 11/27/2023).

[33] Alissa Cooper. “A survey of query log privacy-enhancing techniques from a policy perspective”. In: *ACM Trans. Web* 2.4 (2008), 19:1–19:27. DOI: 10.1145/1409220.1409222. URL: https://doi.org/10.1145/1409220.1409222.

[34] Henry Corrigan-Gibbs and Bryan Ford. “Dissent: accountable anonymous group messaging”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. Ed. by Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov. ACM, 2010, pp. 340–350. DOI: 10.1145/1866307.1866346. URL: https://doi.org/10.1145/1866307.1866346.

[35] *Cryptle Hack Challenge*. https://enarx.dev/cryptle. Enarx, 2022.

[36] Axel Curmi, Christian Colombo, and Mark Vella. “RV-TEE-Based Trustworthy Secure Shell Deployment: An Empirical Evaluation”. In: *J. Object Technol.* 21.2 (2022), 2:1–15. DOI: 10.5381/jot.2022.21.2.a4. URL: https://doi.org/10.5381/jot.2022.21.2.a4.

[37] G. Danezis, R. Dingledine, and N. Mathewson. “Mixminion: design of a type III anonymous remailer protocol”. In: *2003 Symposium on Security and Privacy, 2003.* 2003, pp. 2–15. DOI: 10.1109/SECPRI.2003.1199323.

[38] George Danezis. “Better Anonymous Communications”. PhD thesis. University of Cambridge, 2004.

[39] Tanel Dettenborn. *Open Virtual Trusted Execution Environment*. eng. Tietotekniikan laitos - Department of Pervasive Computing, 2016.

[40] Vasiliki Diamantopoulou et al. “EU GDPR: Toward a Regulatory Initiative for Deploying a Private Digital Era”. In: *Modern Socio-Technical Perspectives on Privacy*. 2022, pp. 427–448. DOI: 10.1007/978-3-030-82786-1_18. URL: https://doi.org/10.1007/978-3-030-82786-1_18.

[41] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. “Tor: The Second-Generation Onion Router”. In: *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*. Ed. by Matt Blaze. USENIX,

2004, pp. 303–320. URL: http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html.

[42] Josep Domingo-Ferrer, Agusti Solanas, and Jordi Castellà-Roca. “h(k)-private information retrieval from privacy-uncooperative queryable databases.">h(k)-private information retrieval from privacy-uncooperative queryable databases”. In: *Online Inf. Rev.* 33.4 (2009), pp. 720–744. DOI: 10.1108/14684520910985693. URL: https://doi.org/10.1108/14684520910985693.

[43] edgelesssys. *ego*. github repo. Accessed on 2023-07-29, Latest Release in 2023. 2021. URL: https://github.com/edgelesssys/ego.

[44] Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. “The Untapped Potential of Trusted Execution Environments on Mobile Devices”. In: *IEEE Secur. Priv.* 12.4 (2014), pp. 29–37. DOI: 10.1109/MSP.2014.38. URL: https://doi.org/10.1109/MSP.2014.38.

[45] *Enarx Technical Overview*. Enarx, 2019. URL: https://enarx.dev/docs/Technical/Introduction.

[46] European Parliament and of the Council. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. Apr. 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679 (visited on 01/29/2023).

[47] Pascal Felber et al. “CoFeed: privacy-preserving Web search recommendation based on collaborative aggregation of interest feedback”. In: *Softw. Pract. Exp.* 43.10 (2013), pp. 1165–1184. DOI: 10.1002/SPE.1127. URL: https://doi.org/10.1002/spe.1127.

[48] Anne Ferry. “Anonymity: The Literary History of a Word”. eng. In: *New Literary History* 33.2 (2002), pp. 193–214. ISSN: 0028-6087.

[49] Fortanix. *Enclave Development Platform*. Fortanix, 2023. URL: https://www.fortanix.com/platform/runtime-encryption/enclave-development-platform (visited on 2023).

[50] Fortanix. *Rust EDP documentation*. Fortanix, 2023. URL: https://edp.fortanix.com/docs/ (visited on 2023).

[51] fortanix. *rust-sgx*. Accessed on 2023-07-29, Latest Release in 2023. 2019. URL: https://github.com/fortanix/rust-sgx.

[52] Cédric Fournet and Jérémy Planul. “Compiling Information-Flow Security to Minimal Trusted Computing Bases”. In: *Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. Ed. by Gilles Barthe. Vol. 6602. Lecture Notes in Computer Science. Springer, 2011, pp. 216–235. DOI:

10.1007/978-3-642-19718-5_12. URL: https://doi.org/10.1007/978-3-642-19718-5_12.

[53] Freax13. *enarx-exploit*. https://github.com/Freax13/enarx-exploit. 2022.

[54] Tal Garfinkel et al. “Terra: a virtual machine-based platform for trusted computing”. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*. Ed. by Michael L. Scott and Larry L. Peterson. ACM, 2003, pp. 193–206. DOI: 10.1145/945445.945464. URL: https://doi.org/10.1145/945445.945464.

[55] GENODE Operating System Framework. *An Exploration of ARM TrustZone Technology*. 2014. URL: https://genode.org/documentation/articles/trustzone (visited on 2023).

[56] Tim Geppert et al. “Overcoming Cloud Concerns with Trusted Execution Environments? Exploring the Organizational Perception of a Novel Security Technology in Regulated Swiss Companies”. In: *55th Hawaii International Conference on System Sciences, HICSS 2022, Virtual Event / Maui, Hawaii, USA, January 4-7, 2022*. ScholarSpace, 2022, pp. 1–8. URL: http://hdl.handle.net/10125/80164.

[57] Tim Geppert et al. “Trusted Execution Environments: Applications and Organizational Challenges”. In: *Frontiers Comput. Sci.* 4 (2022). DOI: 10.3389/fcomp.2022.930741. URL: https://doi.org/10.3389/fcomp.2022.930741.

[58] GlobalPlatform. *TEE system architecture*. 2011. URL: https://globalplatform.org/resource-publication/introduction-to-trusted-execution-environments/ (visited on 2023).

[59] Ian Goldberg. “A Pseudonymous Communications Infrastructure for the Internet”. PhD thesis. University of California, Berkeley, 2000.

[60] Ian Avrum Goldberg and Eric Brewer. “A Pseudonymous Communications Infrastructure for the Internet”. AAI3001848. PhD thesis. 2000. ISBN: 049310500X.

[61] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. “Hiding Routing Information”. In: *Information Hiding, First International Workshop, Cambridge, UK, May 30 - June 1, 1996, Proceedings*. Ed. by Ross J. Anderson. Vol. 1174. Lecture Notes in Computer Science. Springer, 1996, pp. 137–150. DOI: 10.1007/3-540-61996-8_37. URL: https://doi.org/10.1007/3-540-61996-8_37.

[62] Philippe Golle and Kurt Partridge. “On the Anonymity of Home/Work Location Pairs”. In: *Pervasive Computing, 7th International Conference, Pervasive 2009, Nara, Japan, May 11-14, 2009. Proceedings*. Ed. by Hideyuki Tokuda et al. Vol. 5538. Lecture Notes in Computer Science. Springer, 2009, pp. 390–397. DOI: 10.1007/978-3-642-01516-8_26. URL: https://doi.org/10.1007/978-3-642-01516-8_26.

[63] gramineproject. *Gramine*. 2021. URL: https://github.com/gramineproject/gramine (visited on 2023).

[64] Alexander Hamilton, James Madison, and John Jay. *The Federalist Papers*. Published under the pseudonym "Publius". 1787-1788.

[65] Anikó Hannák et al. “Measuring Personalization of Web Search”. In: *CoRR* abs/1706.05011 (2017). arXiv: 1706.05011. URL: http://arxiv.org/abs/1706.05011.

[66] Tony Hansen and Donald E. Eastlake 3rd. *US Secure Hash Algorithms (SHA and HMAC-SHA)*. RFC 4634. Aug. 2006. DOI: 10.17487/RFC4634. URL: https://www.rfc-editor.org/info/rfc4634.

[67] Liwen He et al. “Dynamic Secure Interconnection for Security Enhancement in Cloud Computing”. In: *Int. J. Comput. Commun. Control* 11.3 (2016), pp. 348–357. DOI: 10.15837/ijccc.2016.3.504. URL: https://doi.org/10.15837/ijccc.2016.3.504.

[68] *Health Insurance Portability and Accountability Act of 1996*. Aug 20, 1996. URL: https://aspe.hhs.gov/reports/health-insurance-portability-accountability-act-1996 (visited on 2023).

[69] James Hendricks and Leendert van Doorn. “Secure bootstrap is not enough: shoring up the trusted computing base”. In: *Proceedings of the 11st ACM SIGOPS European Workshop, Leuven, Belgium, September 19-22, 2004*. Ed. by Yolande Berbers and Miguel Castro. ACM, 2004, p. 11. DOI: 10.1145/1133572.1133600. URL: https://doi.org/10.1145/1133572.1133600.

[70] Osama Hosam and Fan BinYuan. “A Comprehensive Analysis of Trusted Execution Environments”. In: *2022 8th International Conference on Information Technology Trends (ITT)*. 2022, pp. 61–66. DOI: 10.1109/ITT56123.2022.9863962.

[71] *I2P - The Invisible Internet Project*. (Visited on 11/28/2023).

[72] Intel. *What Is Intel® SGX?* Intel, 2015. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html (visited on 2023).

[73] Ghilby Varghese Jaison and Charlse M Varghese. “Privacy Protection in Personalized Web Search using Homomorphic Encryption”. In: 2015. URL: https://api.semanticscholar.org/CorpusID:198991451.

[74] Jin Soo Jang et al. “SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment”. In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. URL: https://www.ndss-symposium.org/ndss2015/secret-secure-channel-between-rich-execution-environment-and-trusted-execution-environment.

[75] Bernard J. Jansen, Lu Zhang, and Anna S. Mattila. “User reactions to search engines logos: investigating brand knowledge of web search engines”. In: *Electron. Commer. Res.* 12.4 (2012), pp. 429–454. DOI: 10.1007/s10660-012-9101-0. URL: https://doi.org/10.1007/s10660-012-9101-0.

[76] Simon Paul Johnson. *Intel® SGX and Side-Channels*. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-and-side-channels.html (visited on 06/22/2023).

[77] Benjamin Moore Jr. *Privacy: Studies in Social and Cultural History*. 3rd. Routledge, 1984. DOI: 10.4324/9781315172071. URL: https://doi.org/10.4324/9781315172071.

[78] Marc Juarez and Vicenç Torra. “DisPA: An Intelligent Agent for Private Web Search”. In: *Advanced Research in Data Privacy*. Ed. by Guillermo Navarro-Arribas and Vicenç Torra. Vol. 567. Studies in Computational Intelligence. Springer, 2015, pp. 389–405. DOI: 10.1007/978-3-319-09885-2_21. URL: https://doi.org/10.1007/978-3-319-09885-2%5C_21.

[79] Arttu Paju Juha Nurmi Muhammad Owais Javed. *Privacy Extension for Search Engines*. Latest release in 2023. 2023. URL: https://gitlab.com/nisec/pese (visited on 06/17/2023).

[80] Lynda Kacha and Abdelhafid Zitouni. “An Overview on Data Security in Cloud Computing”. In: *CoRR* abs/1812.09053 (2018). arXiv: 1812.09053. URL: http://arxiv.org/abs/1812.09053.

[81] Dmitrii Kuvaiskii, Gaurav Kumar, and Mona Vij. “Computation offloading to hardware accelerators in Intel SGX and Gramine Library OS”. In: *CoRR* abs/2203.01813 (2022). DOI: 10.48550/arXiv.2203.01813. arXiv: 2203.01813. URL: https://doi.org/10.48550/arXiv.2203.01813.

[82] Mengyuan Li et al. “CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel”. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 717–732. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/li-mengyuan.

[83] Wenhao Li et al. “Building trusted path on untrusted device drivers for mobile devices”. In: *Asia-Pacific Workshop on Systems, APSys’14, Beijing, China, June 25-26, 2014*. ACM, 2014, 8:1–8:7. DOI: 10.1145/2637166.2637225. URL: https://doi.org/10.1145/2637166.2637225.

[84] Dongtao Liu and Landon P. Cox. “VeriUI: attested login for mobile devices”. In: *15th Workshop on Mobile Computing Systems and Applications, HotMobile ’14, Santa Barbara, CA, USA, February 26-27, 2014*. ACM, 2014, 7:1–7:6. DOI: 10.1145/2565585.2565591. URL: https://doi.org/10.1145/2565585.2565591.

[85] Jingbin Liu, Yu Qin, and Dengguo Feng. “SeRoT: A Secure Runtime System on Trusted Execution Environments”. In: *19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020, Guangzhou, China, December 29, 2020 - January 1, 2021*. Ed. by Guojun Wang et al. IEEE, 2020, pp. 30–37. DOI: 10.1109/TrustCom50675.2020.00018. URL: https://doi.org/10.1109/TrustCom50675.2020.00018.

[86] Weijie Liu et al. “Understanding TEE Containers, Easy to Use? Hard to Trust”. In: *CoRR* abs/2109.01923 (2021). arXiv: 2109.01923. URL: https://arxiv.org/abs/2109.01923.

[87] Karsten Loesing. “Privacy-enhancing Technologies for Private Services”. PhD thesis. University of Bamberg Press, 2009.

[88] Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. “Patterns for Secure Boot and Secure Storage in Computer Systems”. In: *ARES 2010, Fifth International Conference on Availability, Reliability and Security, 15-18 February 2010, Krakow, Poland*. IEEE Computer Society, 2010, pp. 569–573. DOI: 10.1109/ARES.2010.110. URL: https://doi.org/10.1109/ARES.2010.110.

[89] Yaqub M. *Search Engine Statistics 2023: Usage, Users & More*. Updated: 2023-09-23. 2023. URL: https://www.businessdit.com/search-engine-statistics/ (visited on 11/14/2023).

[90] Paulo Maciel et al. “A survey on reliability and availability modeling of edge, fog, and cloud computing”. In: *J. Reliab. Intell. Environ.* 8.3 (2022), pp. 227–245. DOI: 10.1007/s40860-021-00154-1. URL: https://doi.org/10.1007/s40860-021-00154-1.

[91] mamccrea et al. *Open Source Solutions to Build Enclave Applications*. Microsoft, 2023. URL: https://learn.microsoft.com/en-us/azure/confidential-computing/enclave-development-oss (visited on 2023).

[92] Michail Maniatakos. “Privilege escalation attack through address space identifier corruption in untrusted modern processors”. In: *Proceedings of the 8th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2013, 26-28 March, 2013, Abu Dhabi, UAE*. IEEE, 2013, pp. 161–166. DOI: 10.1109/DTIS.2013.6527798. URL: https://doi.org/10.1109/DTIS.2013.6527798.

[93] Sinisa Matetic et al. “BITE: Bitcoin Lightweight Client Privacy using Trusted Execution”. In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. Ed. by Nadia Heninger and Patrick Traynor. USENIX Association, 2019, pp. 783–800. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/matetic.

[94] Sonia Ben Mokhtar et al. “RAC: A Freerider-Resilient, Scalable, Anonymous Communication Protocol”. In: *IEEE 33rd International Conference on Distributed Computing Systems, ICDCS 2013, 8-11 July, 2013, Philadelphia, Pennsylvania, USA*. IEEE Computer Society, 2013, pp. 520–529. DOI: 10.1109/ICDCS.2013.52. URL: https://doi.org/10.1109/ICDCS.2013.52.

[95] Sonia Ben Mokhtar et al. “X-Search: Revisiting Private Web Search using Intel SGX”. In: *CoRR* abs/1805.01742 (2018). arXiv: 1805.01742. URL: http://arxiv.org/abs/1805.01742.

[96] Krishna Giri Narra et al. “Privacy-Preserving Inference in Machine Learning Services Using Trusted Execution Environments”. In: *CoRR* abs/1912.03485 (2019). arXiv: 1912.03485. URL: http://arxiv.org/abs/1912.03485.

[97] Helen Nissenbaum and H. W. C. Daniel. “Trackmenot: Resisting Surveillance in Web Search”. In: 2015.

[98] Juha Nurmi. “Understanding the Usage of Anonymous Onion Services: Empirical Experiments to Study Criminal Activities in the Tor Network”. In: 2019.

[99] Occlum. *Occlum*. Latest release in 2023. 2023. URL: https://github.com/occlum/occlum (visited on 02/11/2023).

[100] Mika Ojakangas. “Polis and Oikos: The Art of Politics in the Greek City-State”. In: *The European Legacy* 25 (Feb. 2020), pp. 1–17. DOI: 10.1080/10848770.2020.1721828.

[101] Paul C. van Oorschot. “Memory Errors and Memory Safety: A Look at Java and Rust”. In: *IEEE Secur. Priv.* 21.3 (2023), pp. 62–68. DOI: 10.1109/MSEC.2023.3249719. URL: https://doi.org/10.1109/MSEC.2023.3249719.

[102] Işıl Öz. *Trusted Execution Inside Secure Enclaves – LFX Mentorship Report*. ELISA, 2022. URL: https://elisa.tech/blog/2023/02/01/trusted-execution-inside-secure-enclaves-lfx-mentorship-report/ (visited on 2023).

[103] Jaswant Pakki et al. “Everything You Ever Wanted to Know About Bitcoin Mixers (But Were Afraid to Ask)”. In: *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part I*. Ed. by Nikita Borisov and Claudia Dıaz. Vol. 12674. Lecture Notes in Computer Science. Springer, 2021, pp. 117–146. DOI: 10.1007/978-3-662-64322-8_6. URL: https://doi.org/10.1007/978-3-662-64322-8_6.

[104] Bryan Parno et al. “Memoir: Practical State Continuity for Protected Modules”. In: *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. IEEE Computer Society, 2011, pp. 379–394. DOI: 10.1109/SP.2011.38. URL: https://doi.org/10.1109/SP.2011.38.

[105] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. “Exploring Widevine for Fun and Profit”. In: *43rd IEEE Security and Privacy, SP Workshops 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 277–288. DOI: 10.1109/SPW54247.2022.9833867. URL: https://doi.org/10.1109/SPW54247.2022.9833867.

[106] Sai Teja Peddinti and Nitesh Saxena. “Web search query privacy: Evaluating query obfuscation and anonymizing networks”. In: *J. Comput. Secur.* 22.1 (2014), pp. 155–199. DOI: 10.3233/JCS-130491. URL: https://doi.org/10.3233/JCS-130491.

[107] Marco Pennacchiotti and Ana-Maria Popescu. “A Machine Learning Approach to Twitter User Classification”. In: *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*. Ed. by Lada A. Adamic, Ricardo Baeza-Yates, and Scott Counts. The AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/2886.

[108] People @EECS - University of California, Berkeley. *Privacy-enhancing technologies for the Internet*. 1997. URL: http://people.eecs.berkeley.edu/~daw/papers/privacy-compcon97-www/privacy-html.html (visited on 11/07/2023).

[109] Albin Petit et al. “PEAS: Private, Efficient and Accurate Web Search”. In: *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*. IEEE, 2015, pp. 571–580. DOI: 10.1109/Trustcom.2015.421. URL: https://doi.org/10.1109/Trustcom.2015.421.

[110] Albin Petit et al. “SimAttack: private web search under fire”. In: *J. Internet Serv. Appl.* 7.1 (2016), 2:1–2:17. DOI: 10.1186/s13174-016-0044-x. URL: https://doi.org/10.1186/s13174-016-0044-x.

[111] Julie Pinkerton. *The History of Bitcoin, the First Cryptocurrency*. 2023-08-07. URL: https://money.usnews.com/investing/articles/the-history-of-bitcoin (visited on 09/11/2023).

[112] Rafael Pires et al. “CYCLOSA: Decentralizing Private Web Search through SGX-Based Browser Extensions”. In: *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*. IEEE Computer Society, 2018, pp. 467–477. DOI: 10.1109/ICDCS.2018.00053. URL: https://doi.org/10.1109/ICDCS.2018.00053.

[113] Anthony J. Podlecki and Oliver Taplin. *Aeschylus*. Encyclopedia Britannica, 13 Oct. 2023. 2023. URL: https://www.britannica.com/biography/Aeschylus-Greek-dramatist.

[114] Sergio Prado. *Introduction to Trusted Execution Environment and ARM’s TrustZone*. 2022. URL: https://sergioprado.blog/introduction-to-trusted-execution-environment-tee-arm-trustzone/ (visited on 07/16/2023).

[115] Ivan Puddu et al. “TEEvil: Identity Lease via Trusted Execution Environments”. In: *CoRR* abs/1903.00449 (2019). arXiv: 1903.00449. URL: http://arxiv.org/abs/1903.00449.

[116] John Regehr and Usit Duongsaa. “Preventing interrupt overload”. In: *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES’05), Chicago, Illinois, USA, June 15-17, 2005*. Ed. by Yunheung Paek and Rajiv Gupta. ACM, 2005, pp. 50–58. DOI: 10.1145/1065910.1065918. URL: https://doi.org/10.1145/1065910.1065918.

[117] *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. Official Journal of the European Union. April 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (visited on 2023).

[118] Roland van Rijswijk-Deij and Erik Poll. “Using Trusted Execution Environments in Two-factor Authentication: comparing approaches”. In: *Open Identity Summit 2013, September 9th - 11th 2013, Kloster Banz, Germany*. Ed. by Detlef Hühnlein

and Heiko Roßnagel. Vol. P-223. LNI. GI, 2013, pp. 20–31. URL: https://dl.gi.de/handle/20.500.12116/17195.

[119] Robert B. McKay. *The Right of Privacy: Emanations and Intimations*. 1965. URL: https://repository.law.umich.edu/cgi/viewcontent.cgi?params=/context/mlr/article/5464/&path_info=/ (visited on 02/26/2023).

[120] Jens Rüdinger and Adolf Finger. “Key dependent operation and algorithm specific complexity of statistical side channel attacks”. In: *2009 International Conference on Telecommunications, ICT 2009, Marrakech, Morocco, May 25-27, 2009*. IEEE, 2009, pp. 208–212. DOI: 10.1109/ICTEL.2009.5158645. URL: https://doi.org/10.1109/ICTEL.2009.5158645.

[121] Mark Russinovich et al. “Toward Confidential Cloud Computing: Extending hardware-enforced cryptographic protection to data while in use”. In: *ACM Queue* 19.1 (2021), pp. 49–76. DOI: 10.1145/3454122.3456125. URL: https://doi.org/10.1145/3454122.3456125.

[122] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. “Trusted Execution Environment: What It is, and What It is Not”. In: *2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*. IEEE, 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357. URL: https://doi.org/10.1109/Trustcom.2015.357.

[123] Jerome H. Saltzer and M. Frans Kaashoek. “Chapter 5 - Enforcing Modularity with Virtualization”. In: *Principles of Computer System Design*. Ed. by Jerome H. Saltzer and M. Frans Kaashoek. San Francisco: Morgan Kaufmann, 2009, pp. 199–297. ISBN: 978-0-12-374957-4. DOI: https://doi.org/10.1016/B978-0-12-374957-4.00014-1. URL: https://www.sciencedirect.com/science/article/pii/B9780123749574000141.

[124] Daniel Sangorrín, Shinya Honda, and Hiroaki Takada. “Integrated Scheduling for a Reliable Dual-OS Monitor”. In: *Journal of Information Processing* 5 (2012), pp. 627–638.

[125] Daniel Sangorrín, Shinya Honda, and Hiroaki Takada. “Reliable Device Sharing Mechanisms for Dual-OS Embedded Trusted Computing”. In: *Trust and Trustworthy Computing*. Ed. by Stefan Katzenbeisser et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 74–91. ISBN: 978-3-642-30921-2.

[126] Nuno Santos et al. “Using ARM trustzone to build a trusted language runtime for mobile applications”. In: *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014*. Ed. by Rajeev Balasubramonian, Al Davis, and Sarita V. Adve. ACM, 2014, pp. 67–80. DOI: 10.1145/2541940.2541949. URL: https://doi.org/10.1145/2541940.2541949.

[127] Moritz Schneider et al. “SoK: Hardware-supported Trusted Execution Environments”. In: *CoRR* abs/2205.12742 (2022). DOI: 10.48550/arXiv.2205.12742. arXiv: 2205.12742. URL: https://doi.org/10.48550/arXiv.2205.12742.

[128] Michael Schwarz and Daniel Gruss. “How Trusted Execution Environments Fuel Research on Microarchitectural Attacks”. In: *IEEE Secur. Priv.* 18.5 (2020), pp. 18–27. DOI: 10.1109/MSEC.2020.2993896. URL: https://doi.org/10.1109/MSEC.2020.2993896.

[129] SCONE. *Technical Summary of SCONE*. https://sconedocs.github.io/technical_summary/. Accessed in 2023. SCONE, 2018.

[130] Carlos Segarra et al. “Using Trusted Execution Environments for Secure Stream Processing of Medical Data - (Case Study Paper)”. In: *Distributed Applications and Interoperable Systems - 19th IFIP WG 6.1 International Conference, DAIS 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings*. Ed. by José Pereira and Laura Ricci. Vol. 11534. Lecture Notes in Computer Science. Springer, 2019, pp. 91–107. DOI: 10.1007/978-3-030-22496-7_6. URL: https://doi.org/10.1007/978-3-030-22496-7%5C_6.

[131] A. Serjantov. “On the anonymity of anonymity systems”. Accessed in 2023. PhD thesis. University of Cambridge, 2004. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-604.pdf.

[132] Eran Shalev. “Ancient Masks, American Fathers: Classical Pseudonyms during the American Revolution and Early Republic”. In: *Journal of the Early Republic* 23.2 (2003), pp. 151–172. DOI: 10.2307/3125034. URL: https://doi.org/10.2307/3125034.

[133] Peter Shea. “Book Review: ’Click: What Millions of People Are Doing Online and Why it Matters’ by Bill Tancer”. In: *eLearn Mag.* 2010.1 (2010), p. 8. DOI: 10.1145/1693041.2139010. URL: https://doi.org/10.1145/1693041.2139010.

[134] Youren Shen et al. “Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX”. In: *CoRR* abs/2001.07450 (2020). arXiv: 2001.07450. URL: https://arxiv.org/abs/2001.07450.

[135] Daniel J. Solove. *Nothing to Hide: The False Tradeoff Between Privacy and Security*. Yale University Press, 2011. URL: https://ssrn.com/abstract=3976770.

[136] Mingshen Sun. *Teaclave: A Universal Secure Computing Platform*. Intel, July 14, 2020. URL: https://www.intel.com/content/dam/www/public/us/en/documents/research/mingshen-sun-teaclave-sgx-community-workshop.pdf (visited on 2023).

[137] Yixin Sun et al. “RAPTOR: Routing Attacks on Privacy in Tor”. In: *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, 2015, pp. 271–286. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sun.

[138] Edgeless Systems. *Welcome to EGo!* Edgeless Systems, 2022. URL: https://docs.edgeless.systems/ego/ (visited on 2023).

[139] Sandeep Tamrakar. “Applications of Trusted Execution Environments (TEEs)”. English. Doctoral thesis. School of Science, 2017, 119 + app. 105. ISBN: 978-952-60-7463-4 (electronic), 978-952-60-7464-1 (printed). URL: http://urn.fi/URN:ISBN:978-952-60-7463-4.

[140] Andrew S. Tanenbaum. *Modern operating systems, 3rd Edition*. Pearson Prentice-Hall, 2009. ISBN: 0138134596. URL: https://www.worldcat.org/oclc/254320777.

[141] The Apache Software Foundation. *APACHE LICENSE, VERSION 2.0*. https://www.apache.org/licenses/LICENSE-2.0. Version 2.0, Accessed in 2023. January 2004.

[142] The Apache Software Foundation. *Apache Teaclave*. 2019. URL: https://teaclave.apache.org/docs/ (visited on 2023).

[143] *The WebAssembly System Interface*. https://github.com/bytecodealliance/wasmtime/blob/main/docs/. Accessed in 2023. Bytecode Alliance.

[144] Chia-che Tsai, Donald E. Porter, and Mona Vij. “Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX”. In: *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017*. Ed. by Dilma Da Silva and Bryan Ford. USENIX Association, 2017, pp. 645–658. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai.

[145] Victor Tsai. *emmc v4.41 and v4.5. architecture for high speed functions and features*. 2010. URL: https://www.jedec.org/sites/default/files/Victor_Tsai.pdf (visited on 2023).

[146] Mohib Ullah, Rafiullah Khan, and Muhammad Arshad Islam. “Poshida II, a Multi Group Distributed Peer to Peer Protocol for Private Web Search”. In: *International Conference on Frontiers of Information Technology, FIT 2016, Islamabad, Pakistan, December 19-21, 2016*. IEEE Computer Society, 2016, pp. 75–80. DOI: 10.1109/FIT.2016.022. URL: https://doi.org/10.1109/FIT.2016.022.

[147] Mohib Ullah et al. “ObSecure Logging (OSLo): A Framework to Protect and Evaluate the Web Search Privacy in Health Care Domain”. In: *J. Medical Imaging Health Informatics* 9.6 (2019), pp. 1181–1190. DOI: 10.1166/JMIHI.2019.2708. URL: https://doi.org/10.1166/jmihi.2019.2708.

[148] Mohib Ullah et al. “Profile Aware ObScure Logging (PaOSLo): A Web Search Privacy-Preserving Protocol to Mitigate Digital Traces”. In: *Secur. Commun. Networks* 2022 (2022), 2109024:1–2109024:13. DOI: 10.1155/2022/2109024. URL: https://doi.org/10.1155/2022/2109024.

[149] United Nations. *Universal Declaration of Human Rights*. 1948. URL: https://www.un.org/en/about-us/universal-declaration-of-human-rights (visited on 10/23/2023).

[150] *Use of AI-based applications*. Published: 2023-03-16, Updated: 2023-04-17. Tampere University. 2023. URL: https://intra.tuni.fi/en/teaching/student-administration/academic-integrity-students-0/use-ai-based-applications.

[151] Yue Wang et al. “A Comprehensive Study of WebAssembly Runtime Bugs”. In: *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023, Taipa, Macao, March 21-24, 2023*. Ed. by Tao Zhang, Xin Xia, and Nicole Novielli. IEEE, 2023, pp. 355–366. DOI: 10.1109/SANER56733.2023.00041. URL: https://doi.org/10.1109/SANER56733.2023.00041.

[152] K.H. Waters. “Juvenal and the Reign of Trajan”. In: *Antichthon* 4 (1970), pp. 62–77. DOI: 10.1017/S0066477400004044.

[153] Ahmad Samer Wazan and Frédéric Cuppens. “Cybersecurity in networking: adaptations, investigation, attacks, and countermeasures”. In: *Ann. des Télécommunications* 78.3-4 (2023), pp. 133–134. DOI: 10.1007/s12243-023-00956-9. URL: https://doi.org/10.1007/s12243-023-00956-9.

[154] Nico Weichbrodt et al. “AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves”. In: *Computer Security – ESORICS 2016*. Ed. by Ioannis Askoxylakis et al. Cham: Springer International Publishing, 2016, pp. 440–457. ISBN: 978-3-319-45744-4.

[155] Peter Wilson et al. “Implementing Embedded Security on Dual-Virtual-CPU Systems”. In: *IEEE Des. Test Comput.* 24.6 (2007), pp. 582–591. DOI: 10.1109/MDT.2007.196. URL: https://doi.org/10.1109/MDT.2007.196.

[156] David Isaac Wolinsky et al. “Dissent in Numbers: Making Strong Anonymity Scale”. In: *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*. Ed. by Chandu Thekkath and Amin Vahdat. USENIX Association, 2012, pp. 179–182. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/wolinsky.

[157] Xiaoyan Yan, Qilin Wu, and Youming Sun. “A Homomorphic Encryption and Privacy Protection Method Based on Blockchain and Edge Computing”. In: *Wirel. Commun. Mob. Comput.* 2020 (2020), 8832341:1–8832341:9. DOI: 10.1155/2020/8832341. URL: https://doi.org/10.1155/2020/8832341.

[158] Yourens. *libos*. 2019. URL: https://github.com/Yourens/libos (visited on 2023).

[159] Lianying Zhao et al. “SoK: Hardware Security Support for Trustworthy Execution”. In: *CoRR* abs/1910.04957 (2019). arXiv: 1910.04957. URL: http://arxiv.org/abs/1910.04957.

[160] Shixuan Zhao et al. “vSGX: Virtualizing SGX Enclaves on AMD SEV”. In: *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 321–336. DOI: 10.1109/SP46214.2022.9833694. URL: https://doi.org/10.1109/SP46214.2022.9833694.

APPENDIX A: APPENDIX

SoK: A Systematic Review of TEE Usage for Developing Trusted Applications

Arttu Paju
Tampere University
Tampere, Finland
arttu.paju@tuni.fi

Muhammad Owais Javed
Tampere University
Tampere, Finland
owais.javed@tuni.fi

Juha Nurmi
Tampere University
Tampere, Finland
juha.nurmi@tuni.fi

Juha Savimäki
Tampere University, Unikie Oy
Tampere, Finland
juha.savimaki@tuni.fi

Brian McGillion
Technology Innovation Institute (TII)
Abu Dhabi, UAE
brian@ssrc.tii.ae

Billy Bob Brumley
Tampere University
Tampere, Finland
billy.brumley@tuni.fi

ABSTRACT

Trusted Execution Environments (TEEs) are a feature of modern central processing units (CPUs) that aim to provide a high assurance, isolated environment in which to run workloads that demand both confidentiality and integrity. Hardware and software components in the CPU isolate workloads, commonly referred to as Trusted Applications (TAs), from the main operating system (OS). This article aims to analyse the TEE ecosystem, determine its usability, and suggest improvements where necessary to make adoption easier.

To better understand TEE usage, we gathered academic and practical examples from a total of 223 references. We summarise the literature and provide a publication timeline, along with insights into the evolution of TEE research and deployment. We categorise TAs into major groups and analyse the tools available to developers. Lastly, we evaluate trusted container projects, test performance, and identify the requirements for migrating applications inside them.

CCS CONCEPTS

• **Security and privacy** → **Trusted computing**; *Software security engineering*; *Domain-specific security and privacy architectures*; • **Software and its engineering** → *Application specific development environments.*

KEYWORDS

Trusted Execution Environment; TEE; Confidential Computing; Privacy and Confidentiality; Usability; Application Security

ACM Reference Format:
Arttu Paju, Muhammad Owais Javed, Juha Nurmi, Juha Savimäki, Brian McGillion, and Billy Bob Brumley. 2023. SoK: A Systematic Review of TEE Usage for Developing Trusted Applications. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3600160.3600169

ARES 2023, August 29–September 01, 2023, Benevento, Italy

ACM ISBN 979-8-4007-0772-8/23/08.
https://doi.org/10.1145/3600160.3600169

1 INTRODUCTION

Often, sensitive data is processed on general-purpose operating systems (OSs) which are prone to compromise due to the large number of complex features and services they support. Typically, when an OS is compromised, the applications and their data are also compromised [61]. For instance, if an adversary takes control of an Internet of Things (IoT) device or a cloud instance, the adversary can also access the processes running there [61, 174].

To help mitigate these risks, modern central processing units (CPUs) support a mode of operation that isolates the applications which manage sensitive data from the rest of the system. These isolated environments generally only support enough functionality to enable the processing of this sensitive data. This reduced functionality leads to less code, hence, a smaller Trusted Computing Base (TCB), which in turn enables us to derive trust in those components. Thus, these modes are generally referred to as Trusted Execution Environments (TEEs).

A TEE is a component of the CPU that comprises both hardware and software features with the aim of ensuring the confidentiality and integrity of the code and data loaded inside. The code that runs inside the TEE is often referred to as a Trusted Application (TA), although it does not have to be a full application in the traditional sense; it may comprise only the parts of a larger application that process sensitive data.

The end user of an application or a system is becoming increasingly aware of the need for security, however, they lack the technical knowledge to make informed decisions. As such, the onus is on the developers and maintainers of software to make the correct choices for the user in the most transparent manner possible. For this reason, we take the software developer's perspective and review TEE software development kits (SDKs) and trusted containers (tcons) in order to determine their usability and, consequently, the likelihood of their adoption by applications. Our research questions (RQs) are:

RQ1. Which use case classification describes TAs?
RQ2. Which SDKs are available for TA development?
RQ3. What types of tcons are available?
RQ4. What are the usability implications of porting existing applications to tcons?

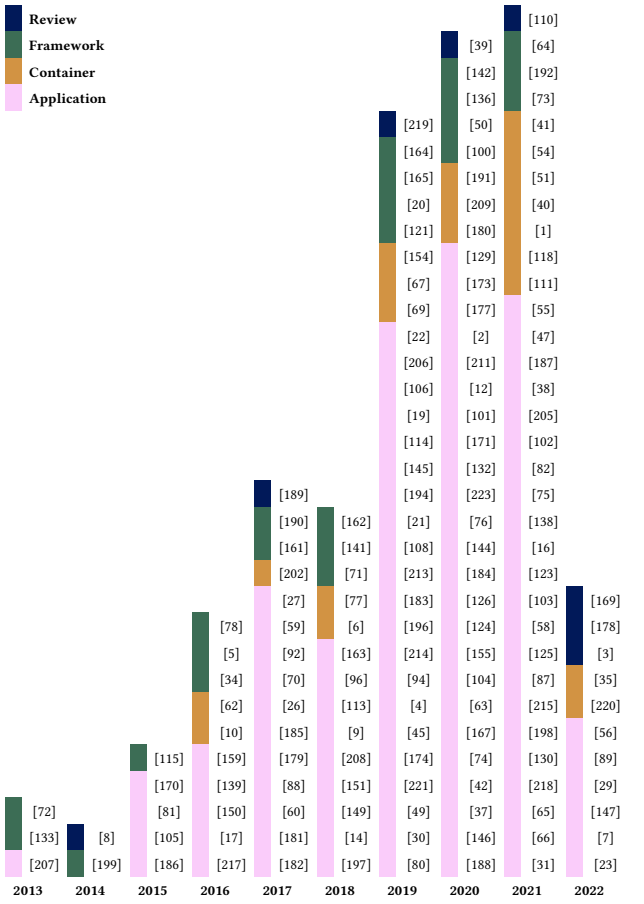

Figure 1: TEE literature falls under the categories of review, framework, container, and application.

TEE implementations are available from a variety of hardware vendors, including AMD Secure Encrypted Virtualization (SEV), Intel Software Guard Extensions (SGX), Intel Trusted Domain Extensions (TDX), ARM TrustZone, and RISC-V Keystone [153]. In addition to TEEs, there are a number of solutions that utilise trusted co-processors: AMD Platform Security Processor (PSP), Google Titan M, and Apple Secure Enclave Processor (SEP) provide many of the same benefits, however, they are discrete from the main CPU.

Cryptographic primitives are utilised extensively to ensure the confidentiality and integrity of the TEE/TA throughout its lifecycle. Attestation of the TEE assures that it is in a known good state before code is loaded; signed binaries ensure that only approved code is loaded; encrypted and integrity-checked data protects it from being read or modified by untrusted parties. All of this is bound to the CPU which protects it from the main OS, potential attackers, and also from the user. As surprising as it may sound, the legitimate user of a system may be considered a potential adversary in certain cases, for example in the case of Digital Rights Management (DRM) or selected banking operations where there is the potential for financial gain by subverting the system's normal operation.

This separation of TEE and Rich Execution Environment (REE) allows for more rigorous verification to be applied to the security-critical parts of a system [8].

The scope of this review is to understand practical TEE deployments. Figure 1 illustrates our references, categorised and ordered by year. In cases of multiple references for the same topic, we list the most peer-reviewed one.

Based on our methodology, we systematically collected 208 of 223 references for this article from 11 March 2022 to 7 June 2022, and only added 15 references after this date. On 13 February 2023, we updated preprint papers with published versions for references that were officially published after our systematic collection of references, but we did not search for additional articles published after that date. Therefore, it is probable that we are missing references published after May 2022.

Our TEE reference search methodology encompasses both academic and applied activity. We discover that the publications fit well under the categories of review, framework, container, and application. The selection of these categories is based on the references' natural fit within them.

We notice that publication velocity increased after 2015. Between 2013 and 2015, there were few publications, but the rate of publication began rising in 2016. We believe this is likely due to the introduction of Intel SGX in 2015, which sparked a surge in interest in TEE-related research. Most ARM TrustZone deployments, which are the most common commercially available TEEs, are proprietary and designed for embedded use cases which renders them inaccessible to most researchers. 19 of the 20 containers listed in Table 6 support Intel SGX hardware. Moreover, 70 of the 103 applications listed in Table 4 support SGX. These facts support the argument that SGX technology is the cause of an increase in publications after 2015.

The number of publications in 2018 remained almost unchanged from the previous year. In 2019, the number of publications nearly doubled and continued to rise in 2020 and 2021. There are more publications in 2021 than ever before, and we speculate this trend will continue in the foreseeable future.

In Figure 1, 103 of the 154 academic publications demonstrate applications. 43 articles cover frameworks and containers. 8 publications are reviews, including surveys and systematisation of knowledge.

This provides the motivation for our article: systematisation of knowledge is required because there are numerous articles but few reviews. These reviews have a limited perspective, typically cover only one piece of hardware, and do not seek real-world usage examples. Our knowledge systematisation assists software developers and encompasses heterogeneous hardware.

This preliminary categorisation of TEE papers shows how academic literature focuses on application demonstrations and helps form a basis for our research questions. It is the initial beginning of our extensive systematisation of knowledge.

Since our survey comprises of 223 references, we wish to highlight some of the most essential ones. In Table 1 and Table 2, we recommend 11 TEE-related articles and 10 repositories, respectively. We base our suggestions on the number of citations for publications and the number of stars for repositories, which we collected

Table 1: Most cited TEE-related articles.

Name	Citations	Stars	Reference
VC3: Trustworthy Data Analytics in the Cloud Using SGX	313	-	[170]
Town Crier: An Authenticated Data Feed for Smart Contracts	281	125	[216, 217]
Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World	160	-	[15]
SCONE: Secure Linux Containers With Intel SGX	123	-	[10, 172]
Keystone: An Open Framework for Architecting Trusted Execution Environments	101	347	[90, 100]
Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX	72	327	[193, 202]
SGX-Log: Securing System Logs With SGX	68	15	[88, 204]
Komodo: Using Verification to Disentangle Secure-Enclave Hardware from Software	67	92	[59, 120]
AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone	58	-	[105]
Sanctum: Minimal Hardware Extensions for Strong Software Isolation	52	49 + 22	[33, 34, 99]
Teechain: A Secure Payment Network with Asynchronous Blockchain Access	52	47	[108, 112]

Table 2: Most starred TEE-related repositories.

Name	Stars	Reference
WebAssembly Micro Runtime	3,425	[22]
MobileCoin	1,102	[129]
Intel Software Guard Extensions for Linux* OS	1,090	[78]
Occlum	1,067	[137, 180]
Teaclave SGX SDK	1,065	[190]
Enarx: Confidential Computing with WebAssembly	1,048	[54]
Asylo	925	[71]
Open Enclave SDK	866	[141]
Teaclave: A Universal Secure Computing Platform	646	[191]
The Confidential Consortium Framework	640	[121]

between 23 January 2023 and 25 January 2023. We collected the number of citations from the *IEEE Xplore*[1], *Springer Link*[2], and *ACM Digital Library*[3] databases. This methodology has limitations when it comes to technical and research papers published elsewhere: the collection method does not take these publications into account. Similarly, *GitHub*[4] alone is utilised to determine the number of stars for each repository.

The chosen articles and repositories give a great overview of some of the most important real-world TEE use cases.

1.1 Related work

As there is prior work on systematising TEE knowledge, we began studying publications and resources that organise TEE utilisation. These data sources cover the following topics.

Software development kits (SDKs). Each CPU vendor has its own TEE. To assist TA development, there are numerous SDKs to aid the software developer [110]. Intel SGX SDK [78], OP-TEE [199], etc. intend to make the development easier.

Trusted containers (tcons). To execute an application within a TEE, a developer must apply framework-specific modifications to the original application, which can be a time-consuming operation. Trusted containers solve this usability issue by allowing direct execution of unmodified binary code within a TEE, or by performing automated transformations on source code prior to loading it into a TEE executable [10]. Certain tcons support multiple hardware backends, eliminating the need for a software developer to make hardware design selections at the code level [110]. We utilise the existing work on tcons by Liu et al. [110] in our categorisations in Table 5 and Table 6. Their work provides a comprehensive analysis of 15 existing tcon solutions' designs and implementations, highlighting the most common security pitfalls. We are not evaluating containers in terms of security, but rather analysing the software wrapper stack and hardware support of 20 containers. Additionally, we check which containers are open source and active as of 2022. We conclude by comparing the active tcons, benchmarking the performance of various tcons, and discussing the usability of the tcons from our perspective.

Applications of TEEs. Tamrakar [189] covers several applications of TEEs, including attestation mechanisms and access control. Our categorisation of TEE utilisation in Table 4 is not based on said work, yet we included the applications presented therein. We also used the study of attestation mechanisms for TEEs by Ménétrey et al. [117] for systemising knowledge of TEE attestation applications. Dangwal et al. [39] explore how TEEs can be used in conjunction with security technologies such as homomorphic encryption and differential privacy for efficient *software-hardware-security* codesign. They propose that security techniques must be combined in order to overcome the inherent limitations of existing technologies.

Curated lists of TEE publications. Schiavoni [168] maintains a curated list of SGX papers while Novella [135] maintains a similar list for TrustZone publications. Whereas the former aims to list all peer-reviewed publications regarding SGX, the latter focuses on attacks against TrustZone-based TEEs and is primarily composed of technical reports, blog postings, and hacking conference presentations.

TEE hardware security. Zhao et al. [219] systemise knowledge of hardware security support for TEEs. Schneider et al. [169] present a systematisation of knowledge pertaining to how various hardware-based TEE solutions meet the security goals of verifiable launch, run-time isolation, trusted I/O, and secure storage. This survey is valuable for understanding how present TEE solution designs achieve their security goals and how existing knowledge can be applied to the development of future TEE solutions.

Attacks against TEEs. There are also other surveys on TEEs not directly relevant to our work. For example, presenting how TEEs reduce the attack surface but do not eliminate it. Numerous attacks have been launched against TEE protection mechanisms and TA implementations [57]. Researchers and practitioners target security flaws and propose solutions for real-world applications, for example, Cerdeira et al. [24] and Koutroumpouchos et al. [95] present a security analysis of popular TrustZone-assisted TEE systems. Akram et al. [3] present a systematisation of knowledge pertaining existing TEEs, highlighting common mechanisms of security guarantees, and offering comparative analyses of different TEE proposals. They also bring up the current limitations of TEEs for high-performance computing systems.

[1] https://ieeexplore.ieee.org/
[2] https://link.springer.com/
[3] https://dl.acm.org/
[4] https://github.com/

1.2 TEE use cases

A TEE technology provides extra protection for various sensitive applications. The following are the most prevalent usage scenarios [189]:

Digital rights management. Copyright holders frequently use TEEs to prevent consumers from copying video or audio [147]. TEEs protect digitally encoded media on connected devices, including smartphones, tablets, and high-definition televisions [11, 53]. Along with the fact that the TEE and the device's display are connected via a protected hardware channel, this prevents the device's owner from reading stored secrets.

Online payments. Mobile wallets, peer-to-peer payments, cryptocurrency wallets, and the use of a mobile device as a point-of-sale terminal – all have well-defined security requirements. Blockchain systems use lightweight clients, which outsource the computational and storage load over full blockchain nodes [114]. It is possible to use TEEs to protect the privacy of the light clients without compromising the performance of the assisting full nodes [114]. TEEs can be used as trusted backend systems to provide the required security to facilitate financial transactions. This may necessitate the entry of a PIN, password, or biometric identifier by the user.

Authentication. TEEs are commonly used to implement biometric identity methods (facial recognition, fingerprint sensor, and voice authorisation). For instance, Android OS can save fingerprint biometrics in the TEE because it is inaccessible and encrypted from the ordinary OS environment [84]. Often, biometric identifications are convenient to use and more difficult to steal than PINs and passwords. TEEs can be utilised to protect the biometric identification method. However, increasingly, biometric data is being stored and verified directly on the sensors and only an attestation is shared with the TEE. Similarly to biometric identification information, cryptographic private keys can also be stored in the TEE. Combining the biometric identification information and the private keys allows passwordless authentication standards such as Apple's passkeys [7].

Trusted cloud. Typically, when a cloud (the server or the backend) is compromised, the adversary gains access to the cloud's processes and data. TEEs provide protection against compromised infrastructure: the adversary is unable to access selective parts of the TA, which safeguards sensitive code and data.

Privacy-preserving data analysis. Machine learning has become an essential part of data processing in several application domains, such as healthcare, stock prediction, and artificial intelligence. Sometimes these applications process sensitive data, and to protect said data, a TEE-based solution can be used to maintain the integrity of the machine learning process and prevent attacks [32].

Runtime integrity. TEEs can be used for runtime integrity, such as real-time kernel protection. If an attacker targets kernel binaries, the security monitoring service can shut down if it is isolated in a secure environment [15].

Secure modular programming. As it decouples functionalities into small, self-contained modules, modular programming is an efficient way to build software architectures for software assets that encourages reuse. In this instance, each module contains everything necessary to perform its intended function, and the TEE permits the execution of the module while protecting it from the vulnerabilities of other modules.

2 METHODOLOGY

2.1 Collecting references for the review

We began our search for scientific literature with *Google Scholar*[5], *arXiv open-access archive*[6], *the DBLP computer science bibliography*[7], *Andor*[8], *ACM Digital Library*, and *IEEE Xplore* using TEE-related search terms, such as "TEE", "Trusted Execution Environment", "OP-TEE", "TrustZone", "(Intel) SGX", "AMD SEV", "confidential computing", etc. While this paints an overall picture of TEE-related scholarly work, it does not cover more applied aspects, such as toolkits and deployments.

To address this gap, we then mined real source code using the Sourcegraph[9] search engine, to find examples of practical TEE utilisation. Sourcegraph covers *GitLab*[10], *GitHub*, and *BitBucket*[11], as well as other public software source repositories. Table 3 details our search terms regarding Sourcegraph, with examples[12]. The most difficult aspect of the mining process was locating appropriate TEE applications, development frameworks, and container repositories. Typically, a keyword search yields thousands of repositories. These repositories contain OSs and kernels, as well as forks and projects with work-in-progress status. Furthermore, we specified "code" as the search type, then sorted and filtered the results to identify the most relevant ones, then finally, manually examined the results.

We based our selection of important phrases on the constants, variables, and functions utilised in the source code of each TEE-based application. The alternative method for picking specific search phrases was to consult the documentation of various TEE-based frameworks and containers, such as the GlobalPlatform API [68]. It reveals applications and other frameworks, containers, and repositories. However, this required combing through each repository manually to obtain the desired results.

2.2 Dimensions for knowledge systematisation

Based on related work and our observations while gathering and reviewing the publications, we organise the TEE literature and practical work.

In Section 3 we address RQ1. Our goal is to assist the reader in comprehending TEEs, how they are utilised, and when, how, and why they could be used. To accomplish this, we tag the applications with 92 distinct keywords, which we merge into 21 primary categories and seven distinct security properties and mechanisms based on initial similarities. We discover that the primary 21 use cases for TEEs in application development are *data analytics*, *cloud*

[5] https://scholar.google.com/
[6] https://arxiv.org/
[7] https://dblp.org/
[8] https://andor.tuni.fi/
[9] https://sourcegraph.com/
[10] https://gitlab.com/
[11] https://bitbucket.org/
[12] https://sourcegraph.com/search?q=context:global+Op-TEE&patternType=literal

Table 3: Using the Sourcegraph search engine, we compile real-world applications of TEEs with the provided search terms.

Search terms	Applications	Containers	Frameworks
SGX_CREATE_ENCLAVE_-EX_PCL_BIT_IDX	1[182]	1[116]	2[141, 190]
TEEC_InvokeCommand	5[106, 127, 163, 177, 206]	0	1[164]
SGX_CREATE_ENCLAVE_EX_-SWITCHLESS	3[28, 93, 176]	0	2[136, 142]
TEEC_MEMREF_TEMP_-OUTPUT	3[36, 128, 157]	0	0
sgx_enclave_id_t	2[22, 129]	1[137]	1[71]
TEEC_RegisterSharedMemory	0	0	1[140, 200]
enarx	2[55, 56]	0	0

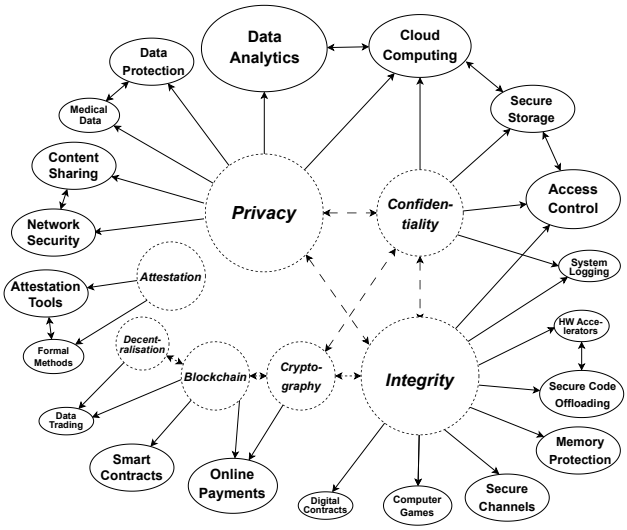

Figure 2: We define which classification aggregates the usage examples after reviewing the TEE example applications. Only the most significant relationships between the categories and the security properties and mechanisms are depicted.

computing, *access control*, *data protection*, *online payments*, *memory protection*, *attestation tools*, *secure storage*, *network security*, *secure channels*, *content sharing*, *secure code offloading*, *smart contracts*, *computer games*, *hardware accelerators*, *formal methods*, *medical data*, *secure system logging*, *web search*, *data trading*, and *digital contracts*. Additionally, we discover that the main security properties and mechanisms related to the use cases are *privacy*, *integrity*, *confidentiality*, *cryptography*, *attestation*, *blockchain*, and *decentralisation*. This is the classification we utilise while reviewing existing TA demonstrations and practical implementations. Figure 2 illustrates our categorisation of the key use cases and the related security properties and mechanisms. Only the strongest relationships are shown in the figure. The size of a category, security property, or mechanism approximately corresponds to its prevalence in existing implementations. Table 4 shows which applications are related to each primary category.

In Section 4 we address RQ2. We compare the software frameworks targeting developers. It is difficult to compare TEE software development tools due to a lack of similar work and public information about their features. Hence, we compile Table 5 detailing the available tools, their supported programming languages, their software licences, the hardware architecture they support, and whether or not they are actively developed as of 2022.

In Section 5 we address RQ3. We organise the tcons for the developers. Again, it is difficult to directly compare tcon tools due to the absence of shared and unique characteristics. In addition, some containers are not actively developed, while others, such as the Enarx container [54], are updated every month with new features. In response, we compile Table 6, which details the available tools, interfaces, software licences, activity of the container project as of 2022, and hardware supported by each tcon.

In Section 6 we address RQ4. We compare the actively developed containers. Additionally, we present Figure 3, which exemplifies a required tcon-specific modification to existing code, demonstrating how challenging it can be to use tcons. Finally, we benchmark the performance of an existing test application using various tcons and present the findings in Table 7.

2.3 Limitations and bias

Lack of documentation of closed-source systems. Companies that own proprietary solutions utilising TEEs typically withhold information about their systems from the public. Therefore, it is difficult to find detailed information regarding closed-source solutions that employ TEEs. Because of this, the data we gathered might be biased towards open-source software and might not show the whole picture of reality. For example, there may be several more closed-source applications and development frameworks for ARM TrustZone than we present in this paper.

Lack of citation data. Certain venues or sources do not disclose the number of citations. This imposes restrictions on which technical and research papers can be listed in Table 1.

Date of initial release. It is often difficult to discover when a specific application, framework, or container was first released. Due to this, the publication year information in Figure 1 may not be entirely accurate.

Manual keyword search. The likelihood of omitting relevant repositories is the most significant shortcoming of a manual search. Although using Sourcegraph as a repository search engine simplifies the search process, it also generates a large number of irrelevant results. There is a chance of missing other applications, development frameworks, and containers that employ different keywords not on our list.

3 APPLICATION SCENARIOS FOR TEE

RQ1: Which use case classification describes TAs?

The TEE isolates and protects the TA code and data in terms of confidentiality and integrity. While we may be unaware, there are a large number of gadgets around us, most notably smartphones, set-top boxes, videogame consoles, and Smart TVs, that utilise a TEE. The number of gadgets utilising a TEE that are designed

for many different purposes results in a wide range of use cases. These use cases vary from everyday user applications to backend services, such as mobile financial services and cloud services [189]. To address RQ1, Table 4 combines TEE application scenarios based on our categorisation.

We gathered a total of 103 application use cases. The categories and the number of references corresponding to each category are the following: *Data analytics* (18), *Cloud computing* (14), *Access control* (14), *Data protection* (10), *Online payments* (8), *Memory protection* (8), *Attestation tools* (7), *Secure storage* (7), *Network security* (7), *Secure channels* (7), *Content sharing* (7), *Secure code offloading* (7), *Smart contracts* (5), *Computer games* (4), *Hardware accelerators* (4), *Formal methods* (3), *Medical data* (3), *Secure system logging* (2), *Web search* (2), *Data trading* (1), and *Digital contracts* (1).

According to Table 4, the vast majority of TEE applications operate on Intel SGX, ARM TrustZone, or both. Only a minority of applications operate on other platforms such as AMD SEV, RISC-V, or GPU TEEs. While most of the references we collected fit within the 21 categories outlined in Section 2, five applications did not fit into any of these categories.

On this basis, the majority of TEE applications aim to provide privacy-preserving data analysis (including machine learning applications). Cloud computing is frequently associated with machine learning applications and is the second-largest TEE usage category in our listing. Application domains surrounding access control, data protection, online payments, and memory protection are also among the most common use cases for TEEs. Albeit noticeably less prevalent than the use cases previously stated, attestation tools, secure storage, network security, secure channels, content sharing, and secure code offloading are all prominent use cases as well with seven references each. Smart contracts, computer games, hardware accelerators, formal methods, and medical data are also fairly prevalent use cases, with three to five references each. The remaining categories represent highly specific TEE use cases with few existing applications. Examples include web search data protection, digital contract signing, and secure system logging.

The number of applications utilising TEEs has steadily increased since 2015. 52 of the 103 references we collected are from 2020 or after, and 48 applications have been deployed to actual users, according to our study. 40 of these 48 applications deployed to actual users are licenced under an open-source licence. Notably, despite this, a large number of proprietary applications with closed-source licences comparable to the Widevine DRM component [147] utilise TEEs. Typically, these proprietary applications are not accompanied by any public documentation or scholarly studies, hence they are largely absent from our work.

4 TOOLS FOR DEVELOPING TEE SOFTWARE

RQ2: Which SDKs are available for TA development?

Numerous middleware frameworks are available to assist developers with TEE development, deployment, and maintenance. To address RQ2, Table 5 combines tools for developing TEE software. In Table 5, we highlight open-source SDKs that are currently being actively developed. Four of the open-source frameworks, such as Webinos [212], are deprecated and no longer under active development. Although there are minor updates, Open-TEE [115] is no longer undergoing substantial development. For our purposes, we consider a project active if there are software updates in 2022, which we assessed on 6 November 2022.

There is a wide selection of frameworks available to software developers for a variety of hardware architectures. The frameworks mentioned are available as open-source software or as brand-focused commercial solutions from various manufacturers, such as the Samsung Knox SDK for Samsung Android devices [162]. 11 of the 23 referred frameworks support Intel SGX, while 13 frameworks support ARM TrustZone, as Table 5 shows. Notably, 21 of the 23 referred frameworks support either Intel SGX or ARM TrustZone, or both.

We researched and compiled a list of supported software languages for active SDK projects. We obtained this information from the SDKs' documentation and examples. This is a limitation, as we can only include supported language information from documented open-source SDKs; thus, these SDKs might have wider non-documented language support. We found that the main languages supported by active SDKs are C and C++. 12 SDKs support at least one of these two languages. Four of them also work with Rust, four work with Java, one supports Go (Edgeless RT), and one supports JavaScript (Confidential Consortium).

The frameworks serve a variety of practical purposes in order to facilitate development efforts. Several frameworks focus on mobile devices and wearables, where the intent is to provide ready-made APIs to support application development [161, 162, 165]. The framework references are also focused on IoT devices or web applications, but due to the wide range of programming language support, the frameworks cover also many other areas [121, 133, 201, 212]. Some of the frameworks are focused on or support very niche areas, like Trustonic's Kinibi-520a SDK [73], where Symmetric-Multi-Processing enables the development of biometric functions like fingerprint scanning and face recognition.

The choice of development framework by the developer is usually severely constrained by the hardware architecture. For example, developers of mobile applications can only use options that are compatible with ARM TrustZone. We find that open-source development frameworks, such as OP-TEE [199], Open Enclave SDK [141], Teaclave TrustZone SDK [192], and Trusty TEE [5] support TrustZone at least in some capacity and are still actively maintained. These frameworks may provide open-source alternatives for mobile application developers, who have traditionally been limited to proprietary closed-source frameworks, such as the Samsung Knox SDK [162] or Trustonic's TEE SDKs [73, 200, 201]. Nevertheless, many open-source frameworks only support specific platforms, so proprietary SDKs may remain the only option for developers on unsupported platforms.

5 TRUSTED CONTAINERS (TCONS)

RQ3: What types of tcons are available?

For an application to function on any TEE technology, the development process must follow framework-specific design solutions. This makes the procedure difficult and time-intensive for application developers. In addition, a developer must implement attestation to trust the application. To address the usability issue with different TEE technologies, a set of tcons enables either the direct execution

Table 4: We classified TEE application scenarios into 21 groups.

		Data analytics	Cloud computing	Access control	Data protection	Online payments	Memory protection	Attestation tool	Secure code offloading	Secure storage	Network security	Secure channels	Content sharing	Smart contracts	Computer games	Emulators	Formal methods	Medical data	Secure system logging	Web search	Digital contracts	Data mining	Arm TrustZone	Intel SGX	GPU TEEs	RISC-V	Real deployments	Open source
2015	AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone	[105]	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	SeCReT: Secure Channel between Rich Execution Environment and TEE	[81]	○	○	○	○	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	TrustOTP: Transforming Smartphones into Secure One-Time Password Tokens	[186]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●
	Using TEEs in Two-factor Authentication: comparing approaches	[207]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	VC3: Trustworthy Data Analytics in the Cloud Using SGX	[170]	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
2016	A Case for Protecting Computer Games With SGX	[17]	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	●	○	○	○
	Oblivious Multi-Party Machine Learning on Trusted Processors	[139]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	Screen after Previous Screens: Spatial-Temporal Recreation of Android App Displays from Memory Images	[36, 159]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	Secure Content-Based Routing Using Intel Software Guard Extensions	[148, 150]	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Town Crier: An Authenticated Data Feed for Smart Contracts	[216, 217]	○	○	●	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
2017	A Formal Foundation for Secure Remote Execution of Enclaves	[185]	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	●	○	●	○	○
	Enhancing Security and Privacy of Tor's Ecosystem by Using TEEs	[86, 92]	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Establishing Mutually Trusted Channels for Remote Sensing Devices with TEEs	[181]	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	IRON: Functional Encryption using Intel SGX	[60]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	●	○	○	○	○
	Komodo: Using verification to disentangle secure-enclave hardware from software	[59, 120]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	●	●
	MIPE: a practical memory integrity protection method in a TEE	[27]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	Private Contact Discovery Service	[182]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Securing Data Analytics on SGX with Randomization	[25, 26]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	SGX-BigMatrix: A Practical Encrypted Data Analytic Framework With Trusted Processors	[179]	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	SGX-Log: Securing System Logs With SGX	[88, 204]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	●	●
	TrustJS: Trusted Client-side Execution of JavaScript	[70]	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
2018	CYCLOSA: Decentralizing Private Web Search through SGX-Based Browser Extensions	[149]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	○	○
	DelegaTEE: Brokered Delegation Using TEEs	[113]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Graviton: TEEs on GPUs	[208]	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○
	LibSEAL: revealing service integrity violations using trusted execution	[14, 97]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Obscuro: A Bitcoin Mixer using TEEs	[18, 197]	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	PubSub-SGX: Exploiting TEEs for Privacy-Preserving Publish/Subscribe Systems	[9, 175]	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	SafeBricks: Shielding Network Functions in the Cloud	[151, 222]	○	●	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	SafeKeeper: Protecting Web Passwords using TEEs	[96, 158]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	TizenFX	[163]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
2019	BITE: Bitcoin Lightweight Client Privacy using Trusted Execution	[114]	○	○	○	○	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Clemmys: towards secure remote execution in FaaS	[194]	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Forward and Backward Private Searchable Encryption with SGX	[4]	○	●	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Fuzzing OP-TEE with AFL	[19, 157]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	Heterogeneous Isolated Execution for Commodity GPUs	[80]	○	○	○	○	○	○	●	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	●
	LightBox: Full-stack Protected Stateful Middlebox at Lightning Speed	[49, 210]	○	○	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	NeXUS: Practical and Secure Access Control on Untrusted Storage Platforms using Client-Side SGX	[44, 45]	○	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	OPERA: Open Remote Attestation for Intel's Secure Enclaves	[30]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	OP-TEE based keymaster and gatekeeper HIDL HAL	[106]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	PrivaTube: Privacy-Preserving Edge-Assisted Video Streaming	[183]	○	○	○	●	○	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	SDK for the Valve Steam Link	[206]	○	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE	[221]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	ShieldStore: Shielded In-memory Key-value Storage with SGX	[93, 94]	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware	[195, 196]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone	[145]	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	Teechain: a secure payment network with asynchronous blockchain access	[107, 108, 112]	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V	[160, 213]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	●
	Trust more, serverless	[21]	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Using TEEs for Secure Stream Processing of Medical Data	[174]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	●	○	○	○	○
	WebAssembly Micro Runtime (WAMR)	[22]	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	ZLiTE: Lightweight Clients for Shielded Zcash Transactions Using Trusted Execution	[214]	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
2020	BDTF: A Blockchain-Based Data Trading Framework with TEE	[184]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	●	○	○	○	○
	BlackMirror: Preventing Wallhacks in 3D Online FPS Games	[146]	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution	[144]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	●	○	○	○	○
	CVShield: Guarding Sensor Data in Connected Vehicle with TEE	[74]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	DarkneTZ: towards model privacy at the edge using TEEs	[126, 127]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Design and Implementation of Hardware-Based Remote Attestation for a Secure Internet of Things	[2]	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	Enabling Rack-scale Confidential Computing using Heterogeneous TEE	[223]	○	○	○	○	○	○	●	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	●
	Fine-Grained Access Control-Enabled Logging Method on ARM TrustZone	[101]	○	○	●	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	GOAT: GPU Outsourcing of Deep Learning Training With Async. Probabilistic Integrity Verification	[12]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Keybuster	[177, 178]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	MobileCoin: Private payments for mobile devices	[129, 134]	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Privacy-preserving Payment Channel Networks using TEE	[104]	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	ProximiTEE: Hardened SGX Attestation by Proximity Verification	[42]	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Reboot-Oriented IoT: Life Cycle Management in TEE for Disposable IoT devices	[188]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	SafeTrace: COVID-19 Self-reporting with Privacy	[173]	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	●	○	○	●	●
	Secure Cloud Storage with Client-side Encryption using a TEE	[37]	○	●	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	secureTF: A Secure TensorFlow Framework	[155]	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	SeGShare: Secure Group File Sharing in the Cloud using Enclaves	[63]	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	SENG, the SGX-Enforcing Network Gateway: Authorizing Communication from Shielded Clients	[171, 176]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Tabellion: secure legal contracts on mobile devices	[124]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	●	○	○	●	●
	Telekine: Secure Computing with Cloud GPUs	[76]	●	●	○	○	○	○	○	●	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○
	Towards Formalization of Enhanced Privacy ID (EPID)-based Remote Attestation in Intel SGX	[167]	○	○	○	○	○	○	●	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	●	○	○	○	○
	TZ4Fabric: Executing Smart Contracts with ARM TrustZone	[131, 132]	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	TZ-MRAS: A Remote Attestation Scheme for the Mobile Terminal Based on ARM TrustZone	[211]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
2021	Atlas: Automated Scale-out of Trust-Oblivious Systems to TEEs	[13, 66]	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Bringing Decentralized Search to Decentralized Services	[103]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	●	●
	Building Enclave-Native Storage Engines for Practical Encrypted Databases	[187]	○	●	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	CURE: A Security Architecture with CUstomizable and Resilient Enclaves	[16]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●
	Distributed Learning in TEE: A Case Study of Federated Learning in SGX	[215]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Enarx Shim SGX	[55]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Formal Verification of a TEE-Based Architecture for IoT Applications	[205]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	IceClave: A TEE for In-Storage Computing	[87]	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	IvyCross: A Trustworthy and Privacy-preserving Framework for Blockchain Interoperability	[102]	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	MeetGo: A TEE for Remote Applications on FPGA	[138]	●	○	○	○	●	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Memory-Efficient Deep Learning Inference in TEEs	[198]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Poster: FLATEE: Federated Learning Across TEEs	[130]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	PPFL: privacy-preserving federated learning with TEEs	[125, 128]	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Privacy-preserving genotype imputation in a TEE	[46, 47]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	●	○	○	○	●
	S2Dedup: SGX-enabled secure deduplication	[122, 123]	○	○	○	●	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	Scalable Memory Protection in the PENGLAI Enclave	[58, 79]	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	ShuffleFL: gradient-preserving federated learning using TEE	[218]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	TEEKAP: Self-Expiring Data Capsule using TEE	[64, 65]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Tora: A Trusted Blockchain Oracle Based on a Decentralized TEE Network	[31, 85]	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	TrustZone-based secure lightweight wallet for hyperledger fabric	[38]	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	TZ-Container: protecting container from untrusted OS with ARM TrustZone	[75]	○	●	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	TZMon: Improving mobile game security with ARM TrustZone	[82, 83]	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	●	●
2022	Exploring Widevine for Fun and Profit	[147]	○	○	●	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●
	MAGE: Mutual Attestation for a Group of Enclaves without Trusted Third Parties	[28, 29]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	●
	MMLedger: A ledger for confidential computing shims for tracking memory management system calls	[56]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	●
	OLIVE: Oblivious and Differentially Private Federated Learning on TEE	[89]	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	Supporting Passkeys	[7]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●
	Toward a Secure, Rich, and Fair Query Service for Light Clients on Public Blockchains	[23]	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○

Table 5: TEE software development tools and language support (●=Yes, ○=No, ·=Not mentioned).

Framework		C	C++	Java	Go	Rust	JavaScript	Intel SGX	AMD SEV	ARM TrustZone	RISC-V	Active (2022)	Open source
Asylo	[71]	·	●	●	·	·	·	●	●	○	○	●	●
Confidential Consortium	[121]	·	●	·	·	·	●	●	○	○	○	●	●
Edgeless RT	[50]	●	●	·	●	●	·	●	○	○	○	●	●
Intel SGX SDK	[78]	·	●	·	·	·	·	●	○	○	○	●	●
Keystone	[90, 100]	●	●	·	·	·	·	○	○	○	●	●	●
Occlum's fork of Intel SGX SDK	[136]	·	·	·	·	·	·	●	○	○	○	●	●
Open-TEE	[115]	·	·	·	·	·	·	●	●	●	○	●	●
OP-TEE	[140, 199]	·	●	·	·	●	·	○	○	●	○	●	●
Open Enclave SDK	[141]	●	●	·	·	·	·	●	○	●	○	●	●
QSEE SDK	[48, 72, 91]	·	·	·	·	·	·	○	○	●	○	○	○
Samsung Knox SDK	[162]	●	·	●	·	·	·	○	○	●	○	●	○
Samsung Knox Tizen SDK	[165]	●	·	●	·	·	·	○	○	●	○	●	○
Samsung mTower	[164]	●	●	·	·	·	·	○	○	●	○	●	●
Samsung TEEGRIS SDK	[161]	●	●	·	·	·	·	○	○	●	○	●	○
Sanctuary	[20, 166]	·	·	·	·	·	·	○	○	●	○	●	○
Sanctum	[33, 34, 99]	·	·	·	·	·	·	○	○	○	●	○	●
SecGear	[142]	●	●	·	·	·	·	●	○	○	○	●	●
Teaclave SGX SDK	[190]	·	·	·	·	·	●	●	○	○	○	●	●
Teaclave TrustZone SDK	[192]	·	·	·	·	·	●	○	○	●	○	●	●
TEEKAP	[64]	·	·	·	·	·	·	●	○	○	○	○	●
Trustonic TEE SDKs	[73, 200, 201]	·	·	·	·	·	·	○	○	●	○	○	○
Trusty TEE	[5]	·	·	●	·	·	·	●	○	●	○	●	●
Webinos	[133, 212]	·	·	·	·	·	·	○	○	●	○	○	●

Table 6: Trusted containers.

Container		libc wrapper	LibOS	WASI	Intel SGX	AMD SEV	Active (2022)	Open source
AccTEE	[43, 69]	○	○	●	●	○	○	●
Anjuna	[6]	○	●	○	●	●	○	○
Apache Teaclave	[191]	○	○	●	●	○	●	●
Chancel	[1]	●	○	○	●	○	○	○
Decentriq	[40]	○	●	○	●	○	○	○
Deflection	[109, 111]	●	○	○	●	○	○	●
EGo SDK	[51]	●	○	○	●	○	●	●
Enarx	[54]	○	○	●	●	●	●	●
Fortanix EDP	[62]	○	●	○	●	○	●	●
GOTEE	[67, 203]	○	○	○	●	○	○	●
Gramine	[193, 202]	○	●	○	●	○	●	●
MesaPy	[119, 209]	○	○	○	●	○	○	●
Mystikos	[41]	○	●	○	●	○	●	●
Occlum	[137, 180]	○	●	○	●	○	●	●
Ratel	[35, 156]	●	○	○	●	○	○	●
Ryoan	[77, 143]	●	○	○	●	○	○	●
SCONE	[10, 172]	●	○	○	●	○	●	●
SGX-LKL	[98, 154]	○	●	○	●	○	○	●
Twine	[116, 118]	○	○	●	●	○	○	●
vSGX	[220]	○	○	○	○	●	●	●

of unmodified binary code inside a TEE or automatic transformation of source code prior to loading it into a TEE executable [110]. In order to address RQ3, Table 6 enumerates tcons.

We collected 20 distinct containers, identified the supported hardware and application middleware interfaces, and determined whether or not the project is open source and active.

17 of the 20 referred tcons are open-source software. If there are software updates in 2022, we consider the tcon project to be active. We evaluated this on 13 October 2022. The open-source tcons saw development activity in the following years: MesaPy (2018); AccTEE (2020); Deflection, GoTEE, Ratel, Ryoan, SGX-LKL, Twine (2021); vSGX, Enarx, Apache Teaclave, EGo SDK, Fortanix EDP, Gramine, Mystikos, and Occlum (2022). Accordingly, there are eight active tcon projects.

We discover that 19 of the 20 tcons support Intel SGX and only three support AMD SEV. In addition, we find no tcons that support TrustZone TEE technology, confining mobile application developers to SDKs. A recent trend seems to be containers that support multiple hardware architectures. The objective is to allow developers to adapt the same program to many platforms without having to alter the source code. Enarx [54] is a good example of such a tcon. Recently published vSGX [220] supports directly running SGX-enabled applications inside AMD SEV.

A system call is an interface between software and the OS through which applications can request services from the OS. Since Intel SGX restricts applications from making system calls, unmodified applications cannot be executed within an enclave.

Seven tcons utilise library OS (LibOS): the missing OS interface that either natively or transparently relays in-enclave system calls to the OS outside the enclave. The LibOS concept predates TEE technologies by at least a decade, motivated by applications in the embedded space due to severe resource constraints [152]. LibOS is an approach to operating system design and implementation where the traditional functionality of an OS is provided by a set of libraries. These libraries are linked directly into the application to create a single address space executable. By encapsulating the operating system functionality within libraries, it becomes easier to define and enforce boundaries between different components, while reducing the TCB. This enables developers to implement security policies at the application level, restricting access to sensitive resources, and preventing unauthorised access to data or interference with other processes. In addition, because the OS primitives are included in the application, this removes the need to invoke system calls and hence, reduces the context switches between user space and kernel space, thus improving performance.

All of this makes LibOS an ideal candidate for use in a TEE, either as a set of standalone applications or as a wrapper around existing applications to reduce the porting effort, e.g., by intercepting system calls from the application and replacing them with LibOS-specific ones.

Six of the 20 tcons utilise wrappers around the C standard library (libc) as an application middleware interface. Executing a system call with libc wrappers, such as EGo SDK, is equivalent to requesting the untrusted OS to perform the corresponding operations outside of the enclave.

The WebAssembly System Interface (WASI) works in a similar fashion and restricts system calls. It provides a runtime for WebAssembly (WASM) binary execution within a TEE [110]. Of the referred 20 tcons, four utilise WASI: AccTEE, Apache Teaclave, Enarx, and Twine execute WASM binaries within a TEE.

From 13 October 2022 to 22 November 2022, we collected and compared the number of Linux system calls against the number of implemented system calls in Gramine, Occlum, Mystikos, Enarx, and Fortanix EDP. For reference, the Linux kernel has a total of 451 system calls, including outdated system calls[13].

Gramine implements 166 system calls[14]. Occlum has 159 implemented system calls[15]. Mystikos implements 102 system calls[16]. Enarx implements the *sallyport*[17] proxying service for service calls and executes WASM binary within a TEE *Keep*. The sallyport protocol implements 31 system calls from a *Keep* to the host[18]. Fortanix EDP – specifically, its user-call API – implements 16 system calls[19], purposefully kept simple to facilitate security audits.

From a security point of view, these tcons increase the TCB. However, if we examine a realistic TA application like machine learning with Python, it has very few system interactions. The developer chooses between implementing the whole software stack from scratch with some SDK or using a tcon; both options have pros and cons.

6 TESTING TCONS

RQ4: What are the usability implications of porting existing applications to tcons?

We compare tcons that are actively in development. A project is deemed active if software updates are released in 2022. Based on the comparison, we deploy and run a benchmark application within the suitable tcons. Realistically, a software developer would choose between the eight active tcon projects based on the functionality they provide.

6.1 Are tcons easy to use?

The trusted containers that use WASM (and Wasmtime) promote the phrase "put your app in a container". When we tested this, we discovered restrictions imposed by WASM that contradict this statement.

While testing WASM containers such as Enarx, we discovered the following obstacles: (1) The selected programming language needs to have native support for WASM development – for instance, Rust. (2) Even with a properly chosen programming language, routine standard library operations like threading and networking may require a redesign of the application. (3) As a result, the majority of Rust's libraries are inoperable by default because they depend on standard libraries. For example, a programmer cannot use existing HTTP libraries to execute an HTTP GET request. (4) Instead, low-level code may be a requirement for even a simple task where you would normally just use one line to call a library. (5) A programmer eventually needs to add *.cargo/config* configurations, macros, and dependencies that are unique to WASM. (6) Development requires mappings between software code and the tcon, for example, pre-opened sockets need to be defined in the *Enarx.toml* configuration file.

```
21 #[tokio::main(flavor = "current_thread")]
22 async fn main() -> io::Result<()> {
23     let listener = {
24         cfg_if::cfg_if! {
25             if #[cfg(not(target_os = "wasi"))] { // Non-WASI
26                 // Create the listening socket
27                 TcpListener::bind("127.0.0.1:12345").await?
28             } else { // WASI-specific workaround
29                 // Enarx.toml defines pre-established socket
30                 let stdlistener = unsafe {
31                     std::net::TcpListener::from_raw_fd(3)
32                 };
33                 stdlistener.set_nonblocking(true).unwrap();
34                 TcpListener::from_std(stdlistener)?
35             }
36         }
37     };
```

Figure 3: This snippet of a TCP proxy application demonstrates WASM and tcon-specific modifications regarding sockets and threading. Line 31 catches the pre-opened socket.

In certain situations, standard libraries need to be replaced with alternatives that support WASM. For instance, *Tokio*[20] is an event-driven, non-blocking I/O platform for developing asynchronous Rust applications, with unstable support for some extra WASM features. However, not all methods are available. For example, new sockets cannot be created from within WASM. Instead, the code must catch the sockets that the tcon creates, as demonstrated by Figure 3.

As a test, we rewrote a TCP proxy application using Tokio with unstable WASM support. In the code, we define WASM build sections with macros and use them to catch the pre-opened sockets. We utilise the "current_thread" macro for threads instead of using a thread pool. Until Wasmtime supports a large number of standard library requirements, it is difficult to simply "put your app in a container".

Using LibOS-based tcons, such as Gramine, Mystikos, and Occlum, we were able to launch diverse applications without modifications. Therefore, it is easier to utilise LibOS-based tcons than those that require WASM.

6.2 Performance of tcons

We test the general-purpose containers Enarx, Gramine, Mystikos, and Occlum to benchmark and execute a Rust application. We select these tcons because they are actively developed, can execute Rust applications, and support Intel SGX. As a comparison, we execute the WASM binary without a secure enclave using Wasmtime. As a second comparison, we test Enarx using AMD SEV hardware instead of Intel SGX hardware.

We forked a paper-based backup scheme application[21] that generates encrypted backups and splits the secret key into multiple key

[13] https://github.com/torvalds/linux/blob/master/include/uapi/asm-generic/unistd.h
[14] https://github.com/gramineproject/gramine/blob/master/libos/include/libos_table.h
[15] https://github.com/occlum/occlum/blob/master/src/libos/src/syscall/mod.rs
[16] https://github.com/deislabs/mystikos/blob/main/kernel/syscall.c
[17] https://github.com/enarx-archive/sallyport
[18] https://github.com/enarx/enarx/blob/main/crates/sallyport/src/host/syscall.rs
[19] https://edp.fortanix.com/docs/api/fortanix_sgx_abi/struct.Usercalls.html
[20] https://tokio.rs/
[21] https://github.com/cyphar/paperback

shards that can be held independently by different users (Shamir's Secret Sharing). We select this application because it is utilised in the real world, is built in Rust, and Rust supports WASM builds. Due to the file interface constraints of tcons, we hard-coded the values in order to test the application.

Table 7: Average application runtimes using Enarx, Gramine, Mystikos, and Occlum. In addition, runtime without TEE as a comparison.

TEE hardware	Software	CPU cycles	App. Runtime (s)
Without TEE	Wasmtime	729,059,936	0.81
AMD SEV	Enarx	1,772,958,278	0.42
Intel SGX	Enarx	8,599,566,984	0.39
	Gramine	1,003,098,599	0.55
	Mystikos	1,380,584,095	0.37
	Occlum	11,702,240,620	2.34

The time to load a container varies greatly, however, we did not compare this metric because it depends on whether the container performs the attestation process. In order to enable attestation, we set up an entire Intel Software Guard Extensions Data Center Attestation Primitives (Intel SGX DCAP)[22].

We measured the real execution time of the application: the main function prints out the duration of the entire code execution. The variance was very low, so the average of 100 samples accurately reflects the execution time and the number of CPU cycles. Notice that CPU cycles take into account the full launch of the tcon and execution of the application, but for the time we measured only the execution time of the application inside the tcon. This means that the number of runtime seconds does not necessarily correlate with the number of CPU cycles. We repeated the Intel SGX tests and Wasmtime tests in the same environment. Unexpectedly, code execution on Occlum takes 2.34 seconds, whereas on Enarx it takes 0.39 seconds while utilising the same WASM binary. There is no obvious reason why Occlum execution is significantly slower. In comparison, without the enclave, the execution time is 0.81 seconds, which is noticeably longer than the Enarx execution time. We used the same build, wasm32-wasi target file, for both execution with Enarx and Wasmtime, yet Enarx is faster, so we suspect that its WASM runtime is lightweight. With Enarx, the application runtime is similar to that on AMD SEV and Intel SGX hardware, but the number of CPU cycles greatly differs. The number of CPU cycles is not comparable in this case because these are different pieces of hardware. As a result, as expected, using a tcon adds overhead to the execution and, when compared to Wasmtime, requires more CPU cycles.

7 CONCLUSION

This article organises TEE applications, frameworks, containers, and reviews in order to determine historic use and usability factors. Our key conclusions are as follows:

[22]https://www.intel.com/content/www/us/en/developer/articles/guide/intel-software-guard-extensions-data-center-attestation-primitives-quick-install-guide.html

Open-source TEE SDKs help TA creation. Typically, a developer must make laborious framework-specific modifications to the original application in order for it to run within a TEE. We listed 23 SDKs available to aid developers with TEE deployment, out of which 17 are open-source software.

Open-source tcons are gaining popularity. A trusted container (tcon) solves the usability issue raised in the previous paragraph by enabling either the direct execution of unmodified binary code within a TEE or the automatic transformation of source code prior to loading a TEE executable. We provided a list of 20 tcons that eliminate the need for software developers to use specific SDKs to write TEE-related code, out of which 17 are open-source software.

Current tcons are not as easy to use as advertised. Our experiments indicate that tcons are not as simple to utilise as advertised. Particularly WASM-based tcons impose strict limitations, necessitating a rewrite of the software's source code and the creation of separate configuration files. In addition, the application must be written in a language that supports WASM natively.

We benchmarked tcons. We benchmarked Enarx, Gramine, Mystikos, and Occlum tcons with the Intel SGX backend. As a comparison, we also ran a WASM binary without a TEE using Wasmtime and Enarx with an AMD SEV backend. Using the identical WASM binary, code execution varied from 2.34 seconds with Occlum to 0.39 seconds with Enarx. According to the measurements, tcons add overhead to the execution and need 1.4 to 16 times more CPU cycles than Wasmtime.

Intel SGX and ARM TrustZone are the most researched. Most of the publications demonstrate application use cases, and Intel SGX is the most popular hardware for applications. In fact, 93 out of 103 TEE applications utilise either Intel SGX, ARM TrustZone, or both. Only a small number of applications can run on other platforms such as AMD SEV, RISC-V, or GPU TEEs.

Current tcons support primarily Intel SGX or AMD SEV. The choice of SDK and tcon by the developer is severely constrained by the hardware architecture. For instance, mobile application developers are restricted to options that are compatible with ARM TrustZone, which means there are no tcons available and a limited number of SDKs to choose from, the majority of which are closed-source frameworks. Some recent tcons, such as Enarx [54] and vSGX [220], enable the execution of the same application within TEEs based on multiple hardware architectures without requiring source code modifications (in theory). Typically, though, tcons only support Intel SGX, AMD SEV, or both.

Data analytics is the most common application category for open-source TAs. Additionally, we examined the primary elements of the execution and the data people attempt to secure with their TAs. We gathered a total of 103 application use cases in Table 4. *Data analytics* (18 references), *Cloud computing* (14 references), and *Access control* (14 references) are the most common of the 21 primary drivers to use TEE.

RISC-V, Sanctum, and Keystone. According to academic references, RISC-V TEE technologies are *interesting*, but few publications are available about them. The scientific community is ideally suited

to pursue the objective of open-source hardware, which is undeniably a concrete development step. The objective is to create a secure and trustworthy hardware-backed enclave for RISC-V. Sanctum [34] and Keystone [100] are seminal steps in this direction, yet we are unaware of any deployments. This lack of mainstream hardware inhibits the growth of the surrounding software ecosystem, somewhat analogous to TrustZone-based TEE technologies, such as On-board Credentials (ObC) [52] in the 2000s: it is clear that ObC predates unified TEE software architectures, such as the GlobalPlatform API [68], yet such standardisation and unification efforts arguably emerged too late to prevent fragmentation of the software ecosystem. In summary, as a community, we should steer TA software development in a consistent and narrow fashion, and to achieve this, we need mainstream hardware available with TEE-relevant hardware-assisted security features that are open source and accessible to developers.

8 FUTURE WORK

RISC-V and applying lessons learned. Several factors have negatively influenced the adoption of TEEs in the past. Moreover, we identified a key research topic based on the analysis of real-world threats and effective mitigation techniques that are most relevant for TEE implementations. With RISC-V as an emerging technology, the standardisation of TEE mechanisms for RISC-V is an excellent opportunity to not only apply valuable lessons learned but to drive the development toward a secure and useable TEE.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952622 (SPIRS), and grant agreement No 804476 (SCARE). Supported in part by the Cybersecurity Research Award granted by the Technology Innovation Institute (TII) in UAE and Technology Innovation Institute's Secure Systems Research Center (SSRC) in UAE.

REFERENCES

[1] Adil Ahmad, Juhee Kim, Jaebaek Seo, Insik Shin, Pedro Fonseca, and Byoungyoung Lee. 2021. CHANCEL: Efficient Multi-client Isolation Under Adversarial Programs. In *NDSS*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/chancel-efficient-multi-client-isolation-under-adversarial-programs/

[2] Jaehwan Ahn, Il-Gu Lee, and Myungchul Kim. 2020. Design and Implementation of Hardware-Based Remote Attestation for a Secure Internet of Things. *Wirel. Pers. Commun.* 114, 1 (2020), 295–327. https://doi.org/10.1007/s11277-020-07364-5

[3] Ayaz Akram, Venkatesh Akella, Sean Peisert, and Jason Lowe-Power. 2022. SoK: Limitations of Confidential Computing via TEEs for High-Performance Compute Systems. In *SEED*. IEEE, 121–132. https://doi.org/10.1109/SEED55351.2022.00018

[4] Ghous Amjad, Seny Kamara, and Tarik Moataz. 2019. Forward and Backward Private Searchable Encryption with SGX. In *EuroSec*. ACM, 4:1–4:6. https://doi.org/10.1145/3301417.3312496

[5] AndroidOpen Source Project. 2016. Trusty TEE. https://source.android.com/security/trusty. Latest rel. 2020.

[6] Anjuna. 2018. Anjuna Confidential Cloud Software. https://www.anjuna.io/ Latest rel. 2022.

[7] ApplePasskeys. 2022. Supporting Passkeys. https://developer.apple.com/documentation/authenticationservices/public-private_key_authentication/supporting_passkeys.

[8] Ghada Arfaoui, Said Gharout, and Jacques Traoré. 2014. Trusted Execution Environments: A Look under the Hood. In *MobileCloud*. IEEE Computer Society, 259–266. https://doi.org/10.1109/MobileCloud.2014.47

[9] Sergei Arnautov, Andrey Brito, Pascal Felber, Christof Fetzer, Franz Gregor, Robert Krahn, Wojciech Ozga, André Martin, Valerio Schiavoni, Fábio Silva, Marcus Tenorio, and Nikolaus Thummel. 2018. PubSub-SGX: Exploiting Trusted Execution Environments for Privacy-Preserving Publish/Subscribe Systems. In *SRDS*. IEEE Computer Society, 123–132. https://doi.org/10.1109/SRDS.2018.00023

[10] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, André Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *OSDI*. USENIX Association, 689–703. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov

[11] N. Asokan. 2019. Hardware-assisted Trusted Execution Environments: Look Back, Look Ahead. In *ACM CCS*. ACM, 1687. https://doi.org/10.1145/3319535.3364969

[12] Aref Asvadishirehjini, Murat Kantarcioglu, and Bradley A. Malin. 2020. GOAT: GPU Outsourcing of Deep Learning Training With Asynchronous Probabilistic Integrity Verification Inside Trusted Execution Environment. *CoRR* abs/2010.08855 (2020). https://arxiv.org/abs/2010.08855

[13] Atlas Runtime. 2022. Atlas: Automated Scale-out of Trust-Oblivious Systems to Trusted Execution Environments. https://github.com/atlas-runtime/applications/

[14] Pierre-Louis Aublin, Florian Kelbert, Dan O'Keeffe, Divya Muthukumaran, Christian Priebe, Joshua Lind, Robert Krahn, Christof Fetzer, David M. Eyers, and Peter R. Pietzuch. 2018. LibSEAL: revealing service integrity violations using trusted execution. In *EuroSys*. ACM, 24:1–24:15. https://doi.org/10.1145/3190508.3190547

[15] Ahmed M. Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. 2014. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World. In *ACM CCS*. ACM, 90–102. https://doi.org/10.1145/2660267.2660350

[16] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In *USENIX Sec.* USENIX Association, 1073–1090. https://www.usenix.org/conference/usenixsecurity21/presentation/bahmani

[17] Erick Bauman and Zhiqiang Lin. 2016. A Case for Protecting Computer Games With SGX. In *SysTEX*. ACM, 4:1–4:6. https://doi.org/10.1145/3007788.3007792

[18] BitObscuro. 2020. Obscuro. https://github.com/BitObscuro/Obscuro

[19] Martijn Bogaard. 2019. Fuzzing OP-TEE with AFL. https://static.linaro.org/connect/san19/presentations/san19-225.pdf.

[20] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. SANCTUARY: ARMing TrustZone with User-space Enclaves. In *NDSS*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/sanctuary-arming-trustzone-with-user-space-enclaves/

[21] Stefan Brenner and Rüdiger Kapitza. 2019. Trust more, serverless. In *SYSTOR*. ACM, 33–43. https://doi.org/10.1145/3319647.3325825

[22] Bytecode Alliance. 2019. WebAssembly Micro Runtime (WAMR). https://github.com/bytecodealliance/wasm-micro-runtime Latest rel. 2022.

[23] Chengjun Cai, Lei Xu, Anxin Zhou, and Cong Wang. 2022. Toward a Secure, Rich, and Fair Query Service for Light Clients on Public Blockchains. *IEEE Trans. Dependable Secur. Comput.* 19, 6 (2022), 3640–3655. https://doi.org/10.1109/TDSC.2021.3103382

[24] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. 2020. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In *IEEE S&P*. IEEE, 1416–1432. https://doi.org/10.1109/SP40000.2020.00061

[25] Swarup Chandra. 2017. Securing Data Analytics on SGX with Randomization. https://github.com/swarupchandra/secure-analytics-sgx

[26] Swarup Chandra, Vishal Karande, Zhiqiang Lin, Latifur Khan, Murat Kantarcioglu, and Bhavani M. Thuraisingham. 2017. Securing Data Analytics on SGX with Randomization. In *ESORICS (LNCS, Vol. 10492)*. Springer, 352–369. https://doi.org/10.1007/978-3-319-66402-6_21

[27] Rui Chang, Liehui Jiang, Wenzhi Chen, Yang Xiang, Yuxia Cheng, and Abdulhameed Alelaiwi. 2017. MIPE: a practical memory integrity protection method in a trusted execution environment. *Clust. Comput.* 20, 2 (2017), 1075–1087. https://doi.org/10.1007/s10586-017-0833-4

[28] Guoxing Chen. 2019. MAGE: Mutual Attestation for a Group of Enclaves without Trusted Third Parties. https://github.com/donnod/linux-sgx-mage Latest rel. 2021.

[29] Guoxing Chen and Yinqian Zhang. 2022. MAGE: Mutual Attestation for a Group of Enclaves without Trusted Third Parties. In *USENIX Sec.* USENIX Association, 4095–4110. https://www.usenix.org/conference/usenixsecurity22/presentation/chen-guoxing

[30] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. 2019. OPERA: Open Remote Attestation for Intel's Secure Enclaves. In *ACM CCS*. ACM, 2317–2331. https://doi.org/10.1145/3319535.3354220

[31] Lili Chen, Rui Yuan, and Yubin Xia. 2021. Tora: A Trusted Blockchain Oracle Based on a Decentralized TEE Network. In *JCC*. 28–33. https://doi.org/10.1109/JCC53141.2021.00016

[32] Yu Chen, Fang Luo, Tong Li, Tao Xiang, Zheli Liu, and Jin Li. 2020. A training-integrity privacy-preserving federated learning scheme with trusted execution environment. *Inf. Sci.* 522 (2020), 69–79. https://doi.org/10.1016/j.ins.2020.02.037

[33] V. Costan. 2015. Sanctum. https://github.com/pwnall/sanctum Latest rel. 2019.

[34] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Sec.* USENIX Association, 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[35] Jinhua Cui, Shweta Shinde, Satyaki Sen, Prateek Saxena, and Pinghai Yuan. 2022. Dynamic Binary Translation for SGX Enclaves. *ACM Trans. Priv. Secur.* 25, 4 (2022), 32:1–32:40. https://doi.org/10.1145/3532862

[36] CyFI-Lab-Public. 2016. RetroScope: Android memory forensics framework. https://github.com/CyFI-Lab-Public/RetroScope

[37] Marciano da Rocha, Dalton Cézane Gomes Valadares, Angelo Perkusich, Kyller Costa Gorgônio, Rodrigo Tomaz Pagno, and Newton Carlos Will. 2020. Secure Cloud Storage with Client-side Encryption using a Trusted Execution Environment. In *CLOSER*. SCITEPRESS, 31–43. https://doi.org/10.5220/0009130600310043

[38] Weiqi Dai, Qinyuan Wang, Zeli Wang, Xiaobin Lin, Deqing Zou, and Hai Jin. 2021. TrustZone-based secure lightweight wallet for hyperledger fabric. *J. Parallel Distributed Comput.* 149 (2021), 66–75. https://doi.org/10.1016/j.jpdc.2020.11.001

[39] Deeksha Dangwal, Meghan Cowan, Armin Alaghi, Vincent T. Lee, Brandon Reagen, and Caroline Trippel. 2020. SoK: Opportunities for Software-Hardware-Security Codesign for Next Generation Secure Computing. In *HASP*. ACM, 8:1–8:9. https://doi.org/10.1145/3458903.3458911

[40] Decentriq. 2021. Decentriq. https://www.decentriq.com/ Latest rel. 2022.

[41] deislabs. 2021. Mystikos. https://github.com/deislabs/mystikos Latest rel. 2022.

[42] Aritra Dhar, Ivan Puddu, Kari Kostiainen, and Srdjan Capkun. 2020. ProximiTEE: Hardened SGX Attestation by Proximity Verification. In *CODASPY*. ACM, 5–16. https://doi.org/10.1145/3374664.3375726

[43] Distributed Systems group at IBR, TU Braunschweig. 2020. AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting. https://github.com/ibr-ds/AccTEE

[44] Briand Djoko. 2020. Secure cloud access/usage control using client-side SGX. https://github.com/sporgj/nexus-code

[45] Judicael Briand Djoko, Jack Lange, and Adam J. Lee. 2019. NeXUS: Practical and Secure Access Control on Untrusted Storage Platforms using Client-Side SGX. In *DSN*. IEEE, 401–413. https://doi.org/10.1109/DSN.2019.00049

[46] Ko Dokmai. 2022. SMac: Secure Genotype Imputation in Intel SGX. https://github.com/ndokmai/sgx-genotype-imputation

[47] Natnatee Dokmai, Can Kockan, Kaiyuan Zhu, XiaoFeng Wang, S. Cenk Sahinalp, and Hyunghoon Cho. 2021. Privacy-preserving genotype imputation in a trusted execution environment. *Cell Systems* 12, 10 (2021), 983–993.e7. https://doi.org/10.1016/j.cels.2021.08.001

[48] David Dong. 2021. Build TA images on different TEE. https://dqdongg.com/android/fingerprint/2021/02/03/Fingerprint-build-ta.html

[49] Huayi Duan, Cong Wang, Xingliang Yuan, Yajin Zhou, Qian Wang, and Kui Ren. 2019. LightBox: Full-stack Protected Stateful Middlebox at Lightning Speed. In *ACM CCS*. ACM, 2351–2367. https://doi.org/10.1145/3319535.3339814

[50] Edgeless Systems. 2020. Edgeless RT. https://github.com/edgelesssys/edgelessrt Latest rel. 2022.

[51] Edgeless Systems. 2021. Welcome to EGo. https://docs.edgeless.systems/ego Latest rel. 2022.

[52] Jan-Erik Ekberg, N. Asokan, Kari Kostiainen, and Aarne Rantala. 2008. Scheduling execution of credentials in constrained secure environments. In *STC*. ACM, 61–70. https://doi.org/10.1145/1456455.1456465

[53] Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. 2014. The Untapped Potential of Trusted Execution Environments on Mobile Devices. *IEEE Secur. Priv.* 12, 4 (2014), 29–37. https://doi.org/10.1109/MSP.2014.38

[54] Enarx. 2021. Enarx. https://github.com/enarx/enarx Latest rel. 2022.

[55] Enarx. 2021. Enarx Shim SGX. https://github.com/enarx/enarx-shim-sgx

[56] Enarx. 2022. MMLedger: A ledger for confidential computing shims for tracking memory management system calls. https://github.com/enarx/mmledger

[57] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Comput. Surv.* 54, 6 (2021), 126:1–126:36. https://doi.org/10.1145/3456631

[58] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2021. Scalable Memory Protection in the PENGLAI Enclave. In *OSDI*. USENIX Association, 275–294. https://www.usenix.org/conference/osdi21/presentation/feng

[59] Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. 2017. Komodo: Using verification to disentangle secure-enclave hardware from software. In *SOSP*. ACM, 287–305. https://doi.org/10.1145/3132747.3132782

[60] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. 2017. IRON: Functional Encryption using Intel SGX. In *ACM CCS*. ACM, 765–782. https://doi.org/10.1145/3133956.3134106

[61] Thomas Fischer, Christian Lesjak, Dominic Pirker, and Christian Steger. 2019. RPC Based Framework for Partitioning IoT Security Software for Trusted Execution Environments. In *IEMCON*. 430–435. https://doi.org/10.1109/IEMCON.2019.8936247

[62] Fortanix. 2016. Fortanix Rust Enclave Development Platform. https://github.com/fortanix/rust-sgx Latest rel. 2022.

[63] Benny Fuhry, Lina Hirschoff, Samuel Koesnadi, and Florian Kerschbaum. 2020. SeGShare: Secure Group File Sharing in the Cloud using Enclaves. In *DSN*. IEEE, 476–488. https://doi.org/10.1109/DSN48063.2020.00061

[64] Mingyuan Gao. 2021. TEEKAP. https://github.com/MingyuanGao/TEEKAP Latest rel. 2022.

[65] Mingyuan Gao, Hung Dang, and Ee-Chien Chang. 2021. TEEKAP: Self-Expiring Data Capsule using Trusted Execution Environment. In *ACSAC*. ACM, 235–247. https://doi.org/10.1145/3485832.3485919

[66] Anagnopoulos Georgios. 2021. *Atlas: Automated Scale-out of Trust-Oblivious Systems to Trusted Execution Environments.* Master's thesis. University of Crete. https://elocus.lib.uoc.gr/dlib/e/6/1/metadata-dlib-1637579552-223704-1365.tkl

[67] Adrien Ghosn, James R. Larus, and Edouard Bugnion. 2019. Secured Routines: Language-based Construction of Trusted Execution Environments. In *USENIX ATC*. USENIX Association, 571–586. https://www.usenix.org/conference/atc19/presentation/ghosn

[68] GlobalPlatform Device Technology. 2010. *TEE Client API Specification Version 1.0.* Technical Report. GlobalPlatform. https://globalplatform.org/wp-content/uploads/2010/07/TEE_Client_API_Specification-V1.0.pdf

[69] David Goltzsche, Manuel Nieke, Thomas Knauth, and Rüdiger Kapitza. 2019. AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting. In *Middleware*. ACM, 123–135. https://doi.org/10.1145/3361525.3361541

[70] David Goltzsche, Colin Wulf, Divya Muthukumaran, Konrad Rieck, Peter R. Pietzuch, and Rüdiger Kapitza. 2017. TrustJS: Trusted Client-side Execution of JavaScript. In *EuroSec*. ACM, 7:1–7:6. https://doi.org/10.1145/3065913.3065917

[71] Google. 2018. Asylo. https://github.com/google/asylo Latest rel. 2022.

[72] Google Git. 2013. qseecom: Add qseecom Driver. https://android.googlesource.com/kernel/msm/+/d316c3dc0464e9703234bc1631700d832b2695bc

[73] Lukas Hanel. 2021. Kinibi-520a: The latest Trustonic Trusted Execution Environment (TEE). https://www.trustonic.com/technical-articles/kinibi-520a-the-latest-trusted-execution-environment-tee/.

[74] Shengtuo Hu, Qi Alfred Chen, Jiwon Joung, Can Carlak, Yiheng Feng, Z. Morley Mao, and Henry X. Liu. 2020. CVShield: Guarding Sensor Data in Connected Vehicle with Trusted Execution Environment. In *AutoSec*. ACM, 1–4. https://doi.org/10.1145/3375706.3380552

[75] Zhichao Hua, Yang Yu, Jinyu Gu, Yubin Xia, Haibo Chen, and Binyu Zang. 2021. TZ-Container: protecting container from untrusted OS with ARM TrustZone. *Sci. China Inf. Sci.* 64, 9 (2021). https://doi.org/10.1007/s11432-019-2707-6

[76] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J. Rossbach, and Emmett Witchel. 2020. Telekine: Secure Computing with Cloud GPUs. In *NSDI*. USENIX Association, 817–833. https://www.usenix.org/conference/nsdi20/presentation/hunt

[77] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2018. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. *ACM Trans. Comput. Syst.* 35, 4 (2018), 13:1–13:32. https://doi.org/10.1145/3231594

[78] Intel Corporation. 2016. Intel(R) Software Guard Extensions for Linux* OS. https://github.com/intel/linux-sgx Latest rel. 2022.

[79] IPADS. 2021. Penglai: Scalable Trusted Execution Environment for RISC-V. https://github.com/Penglai-Enclave/Penglai-Enclave-sPMP

[80] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. 2019. Heterogeneous Isolated Execution for Commodity GPUs. In *ASPLOS*. ACM, 455–468. https://doi.org/10.1145/3297858.3304021

[81] Jin Soo Jang, Sunjune Kong, Minsu Kim, Daegyeong Kim, and Brent ByungHoon Kang. 2015. SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment. In *NDSS*. The Internet Society. https://www.ndss-symposium.org/ndss2015/secret-secure-channel-between-rich-execution-environment-and-trusted-execution-environment

[82] Sanghoon Jeon and Huy Kang Kim. 2021. TZMon: Improving mobile game security with ARM TrustZone. *Comput. Secur.* 109 (2021). https://doi.org/10.1016/j.cose.2021.102391

[83] Sanghoon (Kevin) Jeon. 2018. TZMon: Improving mobile game security with ARM trustzone. https://github.com/kppw99/TZMon

[84] Joel Snyder. 2021. Using biometrics for authentication in Android. https://insights.samsung.com/2021/04/21/using-biometrics-for-authentication-in-android-2

[85] Jseam. 2021. Tora-Zilliqa. https://issueantenna.com/repo/JSeam2/Tora-Zilliqa

[86] kaist-ina. 2019. SGX-Tor. https://github.com/kaist-ina/SGX-Tor

[87] Luyi Kang, Yuqi Xue, Weiwei Jia, Xiaohao Wang, Jongryool Kim, Changhwan Youn, Myeong Joon Kang, Hyung Jin Lim, Bruce L. Jacob, and Jian Huang. 2021. IceClave: A Trusted Execution Environment for In-Storage Computing. In *MICRO*. ACM, 199–211. https://doi.org/10.1145/3466752.3480109

[88] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. 2017. SGX-Log: Securing System Logs With SGX. In *AsiaCCS*. ACM, 19–30. https://doi.org/10.1145/3052973.3053034

[89] Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2022. OLIVE: Oblivious and Differentially Private Federated Learning on Trusted Execution Environment. *CoRR* abs/2202.07165 (2022). https://arxiv.org/abs/2202.07165

[90] Keystone Enclave. 2018. Keystone: An Open-Source Secure Enclave Framework for RISC-V Processors. https://github.com/keystone-enclave/keystone Latest rel. 2022.

[91] Fatima Khalid and Ammar Masood. 2022. Vulnerability analysis of Qualcomm Secure Execution Environment (QSEE). *Comput. Secur.* 116 (2022). https://doi.org/10.1016/j.cose.2022.102628

[92] Seong Min Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. 2017. Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments. In *NSDI*. USENIX Association, 145–161. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kim-seongmin

[93] Taehoon Kim. 2020. ShieldStore. https://github.com/cocoppang/ShieldStore

[94] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. ShieldStore: Shielded In-memory Key-value Storage with SGX. In *EuroSys*. ACM, 14:1–14:15. https://doi.org/10.1145/3302424.3303951

[95] Nikolaos Koutroumpouchos, Christoforos Ntantogian, and Christos Xenakis. 2021. Building Trust for Smart Connected Devices: The Challenges and Pitfalls of TrustZone. *Sensors* 21, 2 (2021), 520. https://doi.org/10.3390/s21020520

[96] Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N. Asokan. 2018. SafeKeeper: Protecting Web Passwords using Trusted Execution Environments. In *WWW*. ACM, 349–358. https://doi.org/10.1145/3178876.3186101

[97] Large-Scale Data & Systems (LSDS) Group. 2021. LibSEAL. https://github.com/lsds/LibSEAL

[98] Large-Scale Data & Systems (LSDS) Group. 2022. SGX-LKL-OE (Open Enclave Edition). https://github.com/lsds/sgx-lkl

[99] Ilia Lebedev. 2019. The MIT Sanctum processor system. https://github.com/ilebedev/sanctum Latest rel. 2020.

[100] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. 2020. Keystone: an open framework for architecting trusted execution environments. In *EuroSys*. ACM, 38:1–38:16. https://doi.org/10.1145/3342195.3387532

[101] Seungho Lee, Hyo Jin Jo, Wonsuk Choi, Hyoseung Kim, Jong Hwan Park, and Dong Hoon Lee. 2020. Fine-Grained Access Control-Enabled Logging Method on ARM TrustZone. *IEEE Access* 8 (2020), 81348–81364. https://doi.org/10.1109/ACCESS.2020.2991431

[102] Ming Li, Jian Weng, Yi Li, Yongdong Wu, Jiasi Weng, Dingcheng Li, and Robert H. Deng. 2021. IvyCross: A Trustworthy and Privacy-preserving Framework for Blockchain Interoperability. *IACR Cryptol. ePrint Arch.* (2021), 1244. https://eprint.iacr.org/2021/1244

[103] Mingyu Li, Jinhao Zhu, Tianxu Zhang, Cheng Tan, Yubin Xia, Sebastian Angel, and Haibo Chen. 2021. Bringing Decentralized Search to Decentralized Services. In *OSDI*. USENIX Association, 331–347. https://www.usenix.org/conference/osdi21/presentation/li

[104] Peng Li, Xiaofei Luo, Toshiaki Miyazaki, and Song Guo. 2020. Privacy-preserving Payment Channel Networks using Trusted Execution Environment. In *ICC*. IEEE, 1–6. https://doi.org/10.1109/ICC40277.2020.9149447

[105] Wenhao Li, Haibo Li, Haibo Chen, and Yubin Xia. 2015. AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone. In *MobiSys*. ACM, 75–88. https://doi.org/10.1145/2742647.2742676

[106] Linaro Security Working Group. 2019. OP-TEE based keymaster and gatekeeper HIDL HAL. https://github.com/linaro-swg/kmgk Latest rel. 2021.

[107] Joshua Lind, Ittay Eyal, Peter R. Pietzuch, and Emin Gün Sirer. 2016. Teechan: Payment Channels Using Trusted Execution Environments. *CoRR* abs/1612.07766 (2016). http://arxiv.org/abs/1612.07766

[108] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter R. Pietzuch. 2019. Teechain: a secure payment network with asynchronous blockchain access. In *SOSP*. ACM, 63–79. https://doi.org/10.1145/3341301.3359627

[109] Weijie Liu. 2021. Deflection (CAT-SGX). https://github.com/StanPlatinum/Deflection

[110] Weijie Liu, Hongbo Chen, XiaoFeng Wang, Zhi Li, Danfeng Zhang, Wenhao Wang, and Haixu Tang. 2021. Understanding TEE Containers, Easy to Use? Hard to Trust. *CoRR* abs/2109.01923 (2021). https://arxiv.org/abs/2109.01923

[111] Weijie Liu, Wenhao Wang, Hongbo Chen, Xiaofeng Wang, Yaosong Lu, Kai Chen, Xinyu Wang, Qintao Shen, Yi Chen, and Haixu Tang. 2021. Practical and Efficient in-Enclave Verification of Privacy Compliance. In *DSN*. IEEE, 413–425. https://doi.org/10.1109/DSN48987.2021.00052

[112] LSDS Group. 2019. Teechain: A Secure Payment Network with Asynchronous Blockchain Access. https://github.com/lsds/Teechain

[113] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. 2018. DelegaTEE: Brokered Delegation Using Trusted Execution Environments. In *USENIX Sec.* USENIX Association, 1387–1403. https://www.usenix.org/conference/usenixsecurity18/presentation/matetic

[114] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. 2019. BITE: Bitcoin Lightweight Client Privacy using Trusted Execution. In *USENIX Sec.* USENIX Association, 783–800. https://www.usenix.org/conference/usenixsecurity19/presentation/matetic

[115] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N. Asokan. 2015. Open-TEE - An Open Virtual Trusted Execution Environment. In *TrustCom*. IEEE, 400–407. https://doi.org/10.1109/Trustcom.2015.400

[116] Jämes Ménétrey. 2022. Twine: An Embedded Trusted Runtime for WebAssembly. https://github.com/JamesMenetrey/unine-twine

[117] Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. 2022. Attestation Mechanisms for Trusted Execution Environments Demystified. In *DAIS (LNCS, Vol. 13272)*. Springer, 95–113. https://doi.org/10.1007/978-3-031-16092-9_7

[118] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. 2021. Twine: An Embedded Trusted Runtime for WebAssembly. In *ICDE*. IEEE, 205–216. https://doi.org/10.1109/ICDE51399.2021.00025

[119] MesaLock Linux. 2019. MesaPy: A Memory-Safe Python Implementation based on PyPy. https://github.com/mesalock-linux/mesapy

[120] Microsoft. 2017. Project Komodo. https://github.com/Microsoft/Komodo

[121] Microsoft. 2019. The Confidential Consortium Framework. https://github.com/microsoft/CCF Latest rel. 2022.

[122] Mariana Miranda. 2021. S2Dedup. https://github.com/mmm97/S2Dedup

[123] Mariana Miranda, Tânia Esteves, Bernardo Portela, and João Paulo. 2021. S2Dedup: SGX-enabled secure deduplication. In *SYSTOR*. ACM, 14:1–14:12. https://doi.org/10.1145/3456727.3463773

[124] Saeed Mirzamohammadi, Yuxin (Myles) Liu, Tianmei Ann Huang, Ardalan Amiri Sani, Sharad Agarwal, and Sung Eun (Summer) Kim. 2020. Tabellion: secure legal contracts on mobile devices. In *MobiSys*. ACM, 220–233. https://doi.org/10.1145/3386901.3389027

[125] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *MobiSys*. ACM, 94–108. https://doi.org/10.1145/3458864.3466628

[126] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *MobiSys*. ACM, 161–174. https://doi.org/10.1145/3386901.3388946

[127] Fan Vincent Mo. 2020. DarkneTZ: Towards Model Privacy at the Edge using Trusted Execution Environments. https://github.com/mofanv/darknetz

[128] Fan Vincent Mo. 2021. Privacy-preserving Federated Learning with Trusted Execution Environments. https://github.com/mofanv/PPFL

[129] MobileCoin Foundation. 2020. MobileCoin: Private payments for mobile devices. https://github.com/mobilecoinfoundation/mobilecoin Latest rel. 2022.

[130] Arup Mondal, Yash More, Ruthu Hulikal Rooparaghunath, and Debayan Gupta. 2021. Poster: FLATEE: Federated Learning Across Trusted Execution Environments. In *EuroS&P*. IEEE, 707–709. https://doi.org/10.1109/EuroSP51992.2021.00054

[131] Christina Müller. 2019. Hyperledger Fabric chaincode execution with OP-TEE. https://github.com/piachristel/open-source-fabric-optee-chaincode

[132] Christina Müller, Marcus Brandenburger, Christian Cachin, Pascal Felber, Christian Göttel, and Valerio Schiavoni. 2020. TZ4Fabric: Executing Smart Contracts with ARM TrustZone. In *SRDS*. IEEE, 31–40. https://doi.org/10.1109/SRDS51746.2020.00011

[133] Cornelius Namiluko, Andrew J. Paverd, and Tulio de Souza. 2013. Towards Enhancing Web Application Security Using Trusted Execution. In *WASH (CEUR Workshop Proceedings, Vol. 1011)*. CEUR-WS.org. http://ceur-ws.org/Vol-1011/4.pdf

[134] Charmaine Ndolo, Sebastian A. Henningsen, and Martin Florian. 2021. Crawling the MobileCoin Quorum System. *CoRR* abs/2111.12364 (2021). https://arxiv.org/abs/2111.12364

[135] Eduardo Novella. 2022. A curated list of public TEE resources for learning how to reverse-engineer and achieve trusted code execution on ARM devices. https://github.com/enovella/TEE-reversing

[136] Occlum team. 2020. Intel(R) Software Guard Extensions for Linux. https://github.com/occlum/linux-sgx Latest rel. 2022.

[137] Occlum team. 2022. Occlum. https://github.com/occlum/occlum

[138] Hyunyoung Oh, Kevin Nam, Seongil Jeon, Yeongpil Cho, and Yunheung Paek. 2021. MeetGo: A Trusted Execution Environment for Remote Applications on FPGA. *IEEE Access* 9 (2021), 51313–51324. https://doi.org/10.1109/ACCESS.2021.3069223

[139] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *USENIX Sec.* USENIX Association, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[140] OP-TEE. 2015. OP-TEE Client API. https://github.com/OP-TEE/optee_client Latest rel. 2022.

[141] Open Enclave. 2018. Open Enclave SDK. https://github.com/openenclave/openenclave Latest rel. 2022.

[142] openEuler. 2020. secGear. https://github.com/openeuler-mirror/secGear Latest rel. 2022.

[143] Operating Systems and Architecture. 2022. Ryoan: A distributed sandbox for untrusted computation on secret data. https://github.com/ut-osa/ryoan

[144] Riccardo Paccagnella, Pubali Datta, Wajih Ul Hassan, Adam Bates, Christopher W. Fletcher, Andrew Miller, and Dave Tian. 2020. Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution. In *NDSS*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/custos-practical-tamper-evident-auditing-of-operating-systems-using-trusted-execution/

[145] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. 2019. StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone. In *USENIX ATC*. USENIX Association, 537–554. https://www.usenix.org/conference/atc19/presentation/park-heejin

[146] Seonghyun Park, Adil Ahmad, and Byoungyoung Lee. 2020. BlackMirror: Preventing Wallhacks in 3D Online FPS Games. In *ACM CCS*. ACM, 987–1000. https://doi.org/10.1145/3372297.3417890

[147] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. 2022. Exploring Widevine for Fun and Profit. In *SP Workshops*. IEEE, 277–288. https://doi.org/10.1109/SPW54247.2022.9833867

[148] Rafael Pires. 2019. Secure content-based routing (SCBR). https://github.com/rafaelppires/scbr

[149] Rafael Pires, David Goltzsche, Sonia Ben Mokhtar, Sara Bouchenak, Antoine Boutet, Pascal Felber, Rüdiger Kapitza, Marcelo Pasin, and Valerio Schiavoni. 2018. CYCLOSA: Decentralizing Private Web Search through SGX-Based Browser Extensions. In *ICDCS*. IEEE Computer Society, 467–477. https://doi.org/10.1109/ICDCS.2018.00053

[150] Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. 2016. Secure Content-Based Routing Using Intel Software Guard Extensions. In *Middleware*. ACM, 1–10. https://doi.org/10.1145/2988336.2988346

[151] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2018. SafeBricks: Shielding Network Functions in the Cloud. In *NSDI*. USENIX Association, 201–216. https://www.usenix.org/conference/nsdi18/presentation/poddar

[152] Donald E. Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C. Hunt. 2011. Rethinking the library OS from the top down. In *ASPLOS*. ACM, 291–304. https://doi.org/10.1145/1950365.1950399

[153] Sergio Prado. 2022. Introduction to Trusted Execution Environment and ARM's TrustZone. https://embeddedbits.org/introduction-to-trusted-execution-environment-tee-arm-trustzone/. Accessed: 2022-06-02.

[154] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, Vasily A. Sartakov, and Peter R. Pietzuch. 2019. SGX-LKL: Securing the Host OS Interface for Trusted Execution. *CoRR* abs/1908.11143 (2019). http://arxiv.org/abs/1908.11143

[155] Do Le Quoc, Franz Gregor, Sergei Arnautov, Roland Kunkel, Pramod Bhatotia, and Christof Fetzer. 2020. securetTF: A Secure TensorFlow Framework. In *Middleware*. ACM, 44–59. https://doi.org/10.1145/3423211.3425687

[156] ratel-enclave. 2022. Ratel - a new framework for instruction-level interposition on enclaved applications. https://github.com/ratel-enclave/ratel

[157] Riscure. 2019. OP-TEE Fuzzer. https://github.com/Riscure/optee_fuzzer Latest rel. 2021.

[158] SafeKeeper. 2018. SafeKeeper - Protecting Web passwords using Trusted Execution Environments. https://github.com/SafeKeeper/safekeeper-server

[159] Brendan Saltaformaggio, Rohit Bhatia, Xiangyu Zhang, Dongyan Xu, and Golden G. Richard III. 2016. Screen after Previous Screens: Spatial-Temporal Recreation of Android App Displays from Memory Images. In *USENIX Sec.* USENIX Association, 1137–1151. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/saltaformaggio

[160] sam1013. 2019. TIMBER-V. https://github.com/sam1013/timberv-riscv-tools/tree/timberv

[161] Samsung. 2017. SAMSUNG TEEGRIS SDK. https://developer.samsung.com/teegris/overview.html

[162] Samsung. 2018. Knox SDK. https://developer.samsungknox.com/knox-sdk. Latest rel. 2022.

[163] Samsung. 2018. TizenFX. https://github.com/Samsung/TizenFX Latest rel. 2022.

[164] Samsung. 2019. mTower. https://github.com/Samsung/mTower Latest rel. 2022.

[165] Samsung. 2019. Welcome to the Knox Tizen SDK for Wearables. https://docs.samsungknox.com/dev/knox-tizen-sdk/index.htm. Latest rel. 2021.

[166] Sanctuary. 2021. Next-Generation Security. Sanctuary. https://sanctuary.dev/en/solutions/security-services/.

[167] Muhammad Usama Sardar, Do Le Quoc, and Christof Fetzer. 2020. Towards Formalization of Enhanced Privacy ID (EPID)-based Remote Attestation in Intel SGX. In *DSD*. IEEE, 604–607. https://doi.org/10.1109/DSD51259.2020.00099

[168] Valerio Schiavoni. 2022. sgx-papers. https://github.com/vschiavoni/sgx-papers

[169] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, and Ronald Perez. 2022. SoK: Hardware-supported Trusted Execution Environments. *CoRR* abs/2205.12742 (2022). https://doi.org/10.48550/arXiv.2205.12742

[170] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *IEEE S&P*. IEEE Computer Society, 38–54. https://doi.org/10.1109/SP.2015.10

[171] Fabian Schwarz and Christian Rossow. 2020. SENG, the SGX-Enforcing Network Gateway: Authorizing Communication from Shielded Clients. In *USENIX Sec.* USENIX Association, 753–770. https://www.usenix.org/conference/usenixsecurity20/presentation/schwarz

[172] Scontain. 2022. SCONE Confidential Computing. https://sconedocs.github.io/

[173] SCRT Labs. 2020. SafeTrace: COVID-19 Self-reporting with Privacy. https://github.com/scrtlabs/SafeTrace

[174] Carlos Segarra, Ricard Delgado-Gonzalo, Mathieu Lemay, Pierre-Louis Aublin, Peter R. Pietzuch, and Valerio Schiavoni. 2019. Using Trusted Execution Environments for Secure Stream Processing of Medical Data. In *DAIS (LNCS, Vol. 11534)*. Springer, 91–107. https://doi.org/10.1007/978-3-030-22496-7_6

[175] SELIS Project. 2019. The SELIS Publish/Subscribe system. https://github.com/selisproject/pubsub

[176] sengsgx. 2020. SENG, the SGX-Enforcing Network Gateway. https://github.com/sengsgx/sengsgx

[177] shakevsky. 2020. Keybuster. https://github.com/shakevsky/keybuster

[178] Alon Shakevsky, Eyal Ronen, and Avishai Wool. 2022. Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design. In *USENIX Sec.* USENIX Association, 251–268. https://www.usenix.org/conference/usenixsecurity22/presentation/shakevsky

[179] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. 2017. SGX-BigMatrix: A Practical Encrypted Data Analytic Framework With Trusted Processors. In *ACM CCS*. ACM, 1211–1228. https://doi.org/10.1145/3133956.3134095

[180] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. 2020. Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX. In *ASPLOS*. ACM, 955–970. https://doi.org/10.1145/3373376.3378469

[181] Carlton Shepherd, Raja Naeem Akram, and Konstantinos Markantonakis. 2017. Establishing Mutually Trusted Channels for Remote Sensing Devices with Trusted Execution Environments. In *ARES*. ACM, 7:1–7:10. https://doi.org/10.1145/3098954.3098971

[182] Signal. 2017. Private Contact Discovery Service. https://github.com/signalapp/ContactDiscoveryService Latest rel. 2022.

[183] Simon Da Silva, Sonia Ben Mokhtar, Stefan Contiu, Daniel Négru, Laurent Réveillère, and Etienne Rivière. 2019. PrivaTube: Privacy-Preserving Edge-Assisted Video Streaming. In *Middleware*. ACM, 189–201. https://doi.org/10.1145/3361525.3361546

[184] Guoxiong Su, Wenyuan Yang, Zhengding Luo, Yinghong Zhang, Zhiqiang Bai, and Yuesheng Zhu. 2020. BDTF: A Blockchain-Based Data Trading Framework with Trusted Execution Environment. In *MSN*. IEEE, 92–97. https://doi.org/10.1109/MSN50589.2020.00030

[185] Pramod Subramanyan, Rohit Sinha, Ilia A. Lebedev, Srinivas Devadas, and Sanjit A. Seshia. 2017. A Formal Foundation for Secure Remote Execution of Enclaves. In *ACM CCS*. ACM, 2435–2450. https://doi.org/10.1145/3133956.3134098

[186] He Sun, Kun Sun, Yuewu Wang, and Jiwu Jing. 2015. TrustOTP: Transforming Smartphones into Secure One-Time Password Tokens. In *ACM CCS*. ACM, 976–988. https://doi.org/10.1145/2810103.2813692

[187] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. 2021. Building Enclave-Native Storage Engines for Practical Encrypted Databases. *Proc. VLDB Endow.* 14, 6 (2021), 1019–1032. http://www.vldb.org/pvldb/vol14/p1019-sun.pdf

[188] Kuniyasu Suzaki, Akira Tsukamoto, Andy Green, and Mohammad Mannan. 2020. Reboot-Oriented IoT: Life Cycle Management in Trusted Execution Environment for Disposable IoT devices. In *ACSAC*. ACM, 428–441. https://doi.org/10.1145/3427228.3427293

[189] Sandeep Tamrakar. 2017. *Applications of Trusted Execution Environments (TEEs)*. Doctoral thesis. Aalto University. http://urn.fi/URN:ISBN:978-952-60-7463-4

[190] The Apache Software Foundation. 2017. Teaclave SGX SDK. https://github.com/apache/incubator-teaclave-sgx-sdk Latest rel. 2022.

[191] The Apache Software Foundation. 2020. Teaclave: A Universal Secure Computing Platform. https://github.com/apache/incubator-teaclave Latest rel. 2022.

[192] The Apache Software Foundation. 2021. Teaclave TrustZone SDK. https://github.com/apache/incubator-teaclave-trustzone-sdk Latest rel. 2022.

[193] The Gramine Project. 2022. Gramine Library OS with Intel SGX Support. https://github.com/gramineproject/gramine

[194] Bohdan Trach, Oleksii Oleksenko, Franz Gregor, Pramod Bhatotia, and Christof Fetzer. 2019. Clemmys: towards secure remote execution in FaaS. In *SYSTOR*. ACM, 44–54. https://doi.org/10.1145/3319647.3325835

[195] Florian Tramer. 2021. SLALOM. https://github.com/ftramer/slalom

[196] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *ICLR*. OpenReview.net. https://openreview.net/forum?id=rJVorjCcKQ
[197] Muoi Tran, Loi Luu, Min Suk Kang, Iddo Bentov, and Prateek Saxena. 2018. Obscuro: A Bitcoin Mixer using Trusted Execution Environments. In *ACSAC*. ACM, 692–701. https://doi.org/10.1145/3274694.3274750
[198] Jean-Baptiste Truong, William Gallagher, Tian Guo, and Robert J. Walls. 2021. Memory-Efficient Deep Learning Inference in Trusted Execution Environments. In *IC2E*. IEEE, 161–167. https://doi.org/10.1109/IC2E52221.2021.00031
[199] TrustedFirmware.org. 2014. OP-TEE Documentation. https://optee.readthedocs.io/en/latest/ Latest rel. 2022.
[200] Trustonic. 2015. Trustonic TEE User Space. https://github.com/Trustonic/trustonic-tee-user-space/
[201] Trustonic. 2018. Secure IoT Development with Kinibi-M. https://www.trustonic.com/technical-articles/kinibi-m/. Latest rel. 2020.
[202] Chia-che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *USENIX ATC*. USENIX Association, 645–658. https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai
[203] USB armory. 2022. GoTEE - example application. https://github.com/usbarmory/GoTEE-example
[204] utds3lab. 2017. SGX-Log: Securing System Logs With SGX. https://github.com/utds3lab/sgx-log
[205] Dalton Cézane Gomes Valadares, Álvaro Alvares de Carvalho César Sobrinho, Angelo Perkusich, and Kyller Costa Gorgônio. 2021. Formal Verification of a Trusted Execution Environment-Based Architecture for IoT Applications. *IEEE Internet Things J.* 8, 23 (2021), 17199–17210. https://doi.org/10.1109/JIOT.2021.3077850
[206] Valve Software. 2019. SDK for the Valve Steam Link. https://github.com/ValveSoftware/steamlink-sdk Latest rel. 2021.
[207] Roland van Rijswijk-Deij and Erik Poll. 2013. Using Trusted Execution Environments in Two-factor Authentication: comparing approaches. In *Open Identity Summit (LNI, Vol. P-223)*. GI, 20–31. https://dl.gi.de/20.500.12116/17195
[208] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *OSDI*. USENIX Association, 681–696. https://www.usenix.org/conference/osdi18/presentation/volos
[209] Huibo Wang, Mingshen Sun, Qian Feng, Pei Wang, Tongxin Li, and Yu Ding. 2020. Towards Memory Safe Python Enclave for Security Sensitive Computation. *CoRR* abs/2005.05996 (2020). https://arxiv.org/abs/2005.05996
[210] Patrick Wang. 2019. LightBox. https://github.com/patrickwang96/LightBox
[211] Ziwang Wang, Yi Zhuang, and Zujia Yan. 2020. TZ-MRAS: A Remote Attestation Scheme for the Mobile Terminal Based on ARM TrustZone. *Secur. Commun. Networks* 2020 (2020), 1756130:1–1756130:16. https://doi.org/10.1155/2020/1756130
[212] webinos. 2013. Secure Web Operating System Application Delivery Environment. https://github.com/webinos/Webinos-Platform
[213] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. 2019. TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V. In *NDSS*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/timber-v-tag-isolated-memory-bringing-fine-grained-enclaves-to-risc-v/
[214] Karl Wüst, Sinisa Matetic, Moritz Schneider, Ian Miers, Kari Kostiainen, and Srdjan Capkun. 2019. ZLiTE: Lightweight Clients for Shielded Zcash Transactions Using Trusted Execution. In *Financial Cryptography (LNCS, Vol. 11598)*. Springer, 179–198. https://doi.org/10.1007/978-3-030-32101-7_12
[215] Tianxing Xu, Konglin Zhu, Artur Andrzejak, and Lin Zhang. 2021. Distributed Learning in Trusted Execution Environment: A Case Study of Federated Learning in SGX. In *IC-NIDC*. IEEE, 450–454. https://doi.org/10.1109/IC-NIDC54101.2021.9660433
[216] Fan Zhang. 2021. Town Crier: An Authenticated Data Feed For Smart Contracts. https://github.com/bl4ck5un/Town-Crier
[217] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town Crier: An Authenticated Data Feed for Smart Contracts. In *ACM CCS*. ACM, 270–282. https://doi.org/10.1145/2976749.2978326
[218] Yuhui Zhang, Zhiwei Wang, Jiangfeng Cao, Rui Hou, and Dan Meng. 2021. ShuffleFL: gradient-preserving federated learning using trusted execution environment. In *CF*. ACM, 161–168. https://doi.org/10.1145/3457388.3458665
[219] Lianying Zhao, He Shuang, Shengjie Xu, Wei Huang, Rongzhen Cui, Pushkar Bettadpur, and David Lie. 2019. SoK: Hardware Security Support for Trustworthy Execution. *CoRR* abs/1910.04957 (2019). http://arxiv.org/abs/1910.04957
[220] Shixuan Zhao, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2022. vSGX: Virtualizing SGX Enclaves on AMD SEV. In *IEEE S&P*. IEEE, 321–336. https://doi.org/10.1109/SP46214.2022.9833694
[221] Shijun Zhao, Qianying Zhang, Yu Qin, Wei Feng, and Dengguo Feng. 2019. SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE. In *ACM CCS*. ACM, 1723–1740. https://doi.org/10.1145/3319535.3363205
[222] Yang Zhou. 2020. SafeBricks. https://github.com/YangZhou1997/SafeBricks
[223] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, and Dan Meng. 2020. Enabling Rack-scale Confidential Computing using Heterogeneous Trusted Execution Environment. In *IEEE S&P*. IEEE, 1450–1465. https://doi.org/10.1109/SP40000.2020.00054

A.1 PESE CODE

```
//****Libraries Declaration****//
use rand::Rng;
use sha2::{Digest, Sha512};
use std::io::{Write, BufReader, BufRead};
use std::net::TcpStream;
use std::time::{SystemTime, UNIX_EPOCH};

//Lightweigh http client for basic http client features
use mini_http;
use mini_http::{Request, Server};

//For Lazy Evaluation Statics: create Static in runtime
use lazy_static::lazy_static;

//For Providing Hashing Algorithm
use std::collections::HashMap;

//For Protecting Shared Data: Helpful in blocking threads
//and wait for the mutex locks to become available
use std::sync::Mutex;

// Application keeps many large HTML pages in RAM memory.
// Reduce the memory usage by compressing the text HTML pages.
use compressed_string::ComprString;

//*****************************//


// Search point address and port for non-wasm 'cargo run'
const ADDRESS: &str = "46.19.38.63"; // IP Address declaration
    Ahmia.fi
const DELAY: u64 = 10; // Delay time in seconds

// Alternative port for Ahmia search, 31000
lazy_static! {
    static ref PORT: Mutex<i32> = {
        Mutex::new(31000)
    };
}

//Struct to hold TCPStream value in connection
```

```
41 struct Stream { connection: TcpStream }
42
43 impl Stream {
44   fn new(connection: TcpStream) -> Self {  //new connection
        establishment
45       Stream { connection: connection }
46   }
47   fn restart(&mut self) {  //restarting connection
48       self.connection = get_tcpstream().unwrap();
49   }
50 }
51
52 lazy_static! {
53     static ref STREAM: Mutex<Stream> = {
54         Mutex::new(Stream::new(get_tcpstream().unwrap()))
55     };
56 }
57
58 //Struct for Webitem account to hold fields like timestamp and
      data
59 struct Webitem { timestamp: u64, data: ComprString }
60
61 impl Webitem { // Save the compressed version of the string
62   fn new(timestamp: u64, data: String) -> Self {
63       Webitem { timestamp: timestamp, data: ComprString::new(&
            data) }
64   }
65   pub fn get_timestamp(&self) -> u64 { //For getting timestamp
66       self.timestamp
67   }
68   pub fn get_data(&self) -> String {  //For getting data
69       self.data.to_string()
70   }
71   pub fn get_size(&self) -> usize {  //for getting size of
        data
72       self.data.compressed_len()
73   }
74 }
75
76 lazy_static! {
77     static ref QUERYMAP: Mutex<HashMap<String, Webitem>> = {
78         Mutex::new(HashMap::new())
```

```
79      };
80  }
81
82  lazy_static! {
83      static ref RESULTMAP: Mutex<HashMap<String, Webitem>> = {
84          Mutex::new(HashMap::new())
85      };
86  }
87
88  //For getting Randomnumber range in between -2147483648 to
       2147483647
89  fn randomnumber(min: i32, max: i32) -> i32 {
90      assert!(min < max);
91      let number: i32 = rand::thread_rng().gen_range(min..max);
92      assert!(min <= number && number < max);
93      return number;
94  }
95
96  // A random number from -2147483648 to 2147483647 - 1, around
       2^32 possibilities
97  lazy_static! {
98      static ref RANDOMNUMBER: Mutex<i32> = {
99          Mutex::new(randomnumber(i32::MIN, i32::MAX))
100     };
101 }
102
103 fn timenow() -> u64 {
104     // Current time: return the total number of seconds
105     let seconds: u64 = SystemTime::now()
106         .duration_since(UNIX_EPOCH)  //Unix timestamp elapsed
               from 1 Jan 1970
107         .unwrap()
108         .as_secs();
109     return seconds;
110 }
111
112 fn random_choice() -> (String, String) {
113     // Selection of Random choice from the hashmap and return
           item.
114     let size: i32 = QUERYMAP.lock().unwrap().len().try_into().
           unwrap();
115     assert!(size > 0); // Size is always larger than 0
```

```
116
117     // Random selection from the hashmap
118     let selection: i32 = randomnumber(0, size); // 0 to size-1
119     assert!(selection < size);
120
121     let mut counter: i32 = 0;
122     loop { // Return the selected item
123         for (key, value) in QUERYMAP.lock().unwrap().iter() {
124             if selection == counter {
125                 let copy_key = format!("{}", key);
126                 let copy_value = format!("{}", value.get_data
                        ());
127                 return (copy_key, copy_value); // Return
                        values
128             }
129             counter += 1;
130             assert!(counter < size);
131         }
132     }
133 }
134
135 fn suffle_select() -> Result<(String, String), (String, String
        )> {
136     // Select of random item and remove it from the hashmap
137     let size = QUERYMAP.lock().unwrap().len();
138
139     if size == 0 { // No items so nothing to return
140         return Err(("".to_string(), "".to_string())); // None
141     }
142
143     let (copy_key, copy_value) = random_choice();
144     QUERYMAP.lock().unwrap().remove(&copy_key);
145     return Ok((copy_key, copy_value));
146 }
147
148 //For generating SHA512 hash of 64-bytes
149 fn hash512(data: &[u8]) -> String {
150     let mut hasher = Sha512::new();
151     hasher.update(data);
152     format!("{:x}", hasher.finalize())
153 }
154
```

```
155 //For generating id from the SHA512 hash (initial 20 bytes)
156 fn id_hash(input: String) -> String {
157     let number = RANDOMNUMBER.lock().unwrap();
158     let str_hash = format!("/result/{}", hash512(format!("{} -
            {:?}", input, number).as_bytes()));
159     return str_hash[0..20].to_string();
160 }
161
162 //Function for Redirecting HTML page to the hash URL
163 fn redirect_html(hash: String) -> String {
164     // Return the redirect HTML page for the hash URL
165     let html_string = format
            !("{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}",
166       "<!DOCTYPE html>",
167       "<html>",
168       "<head>",
169       "<meta http-equiv='refresh' content='",
170       DELAY,
171       "; URL=",
172       hash,
173       "' />",
174       "</head>",
175       "<body>",
176       "<p> Wait seconds: ",
177       DELAY,
178       "</p> <p>You'll be sent directly to the results page.</p
              >",
179       "</body>",
180       "</html>",
181     );
182     return html_string.to_string();
183 }
184
185 #[cfg(target_os = "wasi")]
186 fn get_server() -> Server {
187     mini_http::Server::preopened().unwrap()
188 }
189
190 #[cfg(not(target_os = "wasi"))]
191 fn get_server() -> Server {
192     mini_http::Server::new("127.0.0.1:34455").unwrap()
193 }
```

```
194
195 // When the target of the build is wasi
196 // Making TCP connection with the address
197 #[cfg(target_os = "wasi")]
198 fn get_tcpstream() -> std::result::Result<TcpStream, Box<dyn
    std::error::Error>> {
199     // Use existing TCP connections
200     // Enarx has already established it (ahmif.fi
        :31000...31009)
201     println!("Connect to {}:{}", ADDRESS, PORT.lock().unwrap()
        );
202     // NOTE: This is for wasm: if the original connection to
        the port 31000 is lost
203     let desc = *PORT.lock().unwrap() - 31000 + 4; // The first
         call returns 4
204     use std::os::wasi::io::FromRawFd;
205     let stdstream = unsafe { std::net::TcpStream::from_raw_fd(
        desc) };
206     *PORT.lock().unwrap() += 1; // NOTE: Increase the port for
         the next call
207     Ok(stdstream)
208 }
209
210 // When the target of the build is not wasi
211 // Making TCP connection with the Address
212 #[cfg(not(target_os = "wasi"))]
213 fn get_tcpstream() -> std::result::Result<TcpStream, std::io::
    Error> {
214     println!("Connect to {}:{}", ADDRESS, PORT.lock().unwrap()
        );
215     let searchaddress = format!("{}:{}", ADDRESS, PORT.lock().
        unwrap());
216     std::net::TcpStream::connect(searchaddress)
217 }
218
219 //Function to read content length of http packet
220 fn read_content_length(http: &String) -> i32 {
221     let str_lines: Vec<&str> = http.lines().collect();
222     for line in str_lines{
223         if line.contains("Content-Length: ") {
224             let str_parts: Vec<&str> = line.split_whitespace()
                .collect();
```

```
225             return str_parts[1].parse::<i32>().unwrap();
226         }
227     }
228     return 0;
229 }
230
231 fn htmlpage(page: String) -> String {
232     // Remove redirect search result links to point directly
        to the result pages
233     let part1 = "<a\n                href=\"/search/search/
        redirect?search_term=";
234     let replacement1 = format!("{}{}", "<!--", part1);
235     let part2 = "&redirect_url=";
236     let replacement2 = format!("{}{}", part2, "-->\n
                    <a href=\"");
237     let mut htmlpage = page.replace(part1, &replacement1).
        replace(part2, &replacement2);
238     // Add padding: Each HTML page is around 2 000 000 bytes =
         2MB.
239     let mut hasher = Sha512::new();
240     hasher.update(randomnumber(0, 1000000000).to_string()); //
         Hash a random number
241     let hash_str = format!("{:x}", hasher.finalize()); //
        Format the hash to string
242     // 128 multiplied 1000..2000 times
243     let pad = hash_str.repeat(randomnumber(1000, 2000).
        try_into().unwrap());
244     // Size is 1 700 000 + variance
245     while htmlpage.len() < 1700000 { // Add random nonsense
        and noise to the final size
246         let padding = format!("{}{}{}", "\n    <!-- ", pad, "
            -->    \n</html>");
247         htmlpage = htmlpage.replace("</html>", &padding); //
            Add at least 128 000 bytes
248     }
249     return htmlpage;
250 }
251
252 fn request(key: &String, value: &String) -> std::result::
    Result<String, String> {
253     println!("Try to forward: {}", value);
254     let mut stream = &STREAM.lock().unwrap().connection;
```

```
255     let result_write = stream.write_all(value.as_bytes());
256     match result_write {
257         Ok(_)=> { println!("Forwarded: {}", value); },
258         Err(_)=> { return Err("Failed request".to_string()); }
259     }
260     stream.flush().unwrap(); // Flush or server never
           responses
261     // Receive data from TCP
262     let mut reader = BufReader::new(stream);
263     let mut result = String::new();
264     let mut readbytes: i32 = 0;
265     loop { // Read bytes until no data from the server
266         let bytes = reader.read_line(&mut result).unwrap();
267         if bytes == 0 { break; }
268         if readbytes == 0 && result.contains("\r\n\r\n") {
269             readbytes = read_content_length(&result);
270             result = String::new();
271         }
272         if readbytes > 0 && result.len() >= readbytes.try_into
               ().unwrap() { break };
273     }
274     if result.len() > 0 {
275         println!("Received {:?} bytes", result.len());
276         let page = htmlpage(result.to_string());
277         println!("With added padding {:?} bytes.", page.len())
               ;
278         RESULTMAP.lock().unwrap().insert(key.to_string(),
               Webitem::new(timenow(), page));
279         let size = RESULTMAP.lock().unwrap().get(&key.
               to_string()).unwrap().get_size();
280         println!("Compressed version consumes {} bytes RAM.",
               size);
281         return Ok("Results OK".to_string()); // Done, ready,
               return
282     }
283     else { return Err(format!("Failed: {}", value).to_string()
           ); }
284 }
285
286 //Function for Fetching Results
287 fn get_results(url: String) -> std::result::Result<Vec<u8>,
       String> {
```

```
288     if !QUERYMAP.lock().unwrap().contains_key(&url) && !
           RESULTMAP.lock().unwrap().contains_key(&url) {
289         return Err("Invalid URL".to_string());
290     }
291     // If results already available return the result
292     if RESULTMAP.lock().unwrap().contains_key(&url) {
293         return Ok(RESULTMAP.lock().unwrap().get(&url).unwrap()
               .get_data().as_bytes().to_vec());
294     }
295     // Check that the user waited the delay
296     if QUERYMAP.lock().unwrap().contains_key(&url) {
297         let seconds = QUERYMAP.lock().unwrap().get(&url).
               unwrap().get_timestamp();
298         if timenow() < (seconds + DELAY) {
299             return Ok(redirect_html(url).as_bytes().to_vec());
                   // Wait till delay time is full
300         }
301     }
302     // Else execute all queries
303     while QUERYMAP.lock().unwrap().len() > 0 { // Process all
           the items in random order
304         match suffle_select() { // Random selection which
               removes the item from the hashmap
305             Ok((key, value)) => {
306                 for _i in 0..3 {
307                     match request(&key, &value) {
308                         Ok(_) => { break; },
309                         Err(msg) => { println!("{}", msg);  }
310                     }
311                     STREAM.lock().unwrap().restart(); // Try
                           with a new connection
312                 }
313             },
314             Err(_) => { println!("Error: failed to select
                   random item"); }
315         }
316     }
317     if RESULTMAP.lock().unwrap().contains_key(&url){
318         return Ok(RESULTMAP.lock().unwrap().get(&url).unwrap()
               .get_data().as_bytes().to_vec());
319     }
320     return Err("ERROR: no result".to_string());
```

```
}

//Function for the exection of user search query
fn execute_query(req: &Request) -> Vec<u8> {
    if !req.uri().to_string().contains("/result/") {
        return "Not Found".as_bytes().to_vec();
    }
    let mut bytes_list : Vec<u8> = Vec::new();
    let result = get_results(req.uri().to_string());
    match result {
        Ok(bytes)=>{
            bytes_list.extend(bytes);
        },
        Err(msg)=>{
            println!("Error: {}", msg);
        }
    }
    return bytes_list;
}

//Function for receiving user search query
fn collect_query(req: &Request) -> Vec<u8> {
    println!("Received: {}", req.uri());
    let httpget = format!("{}{}{}{}{}{}",
        "GET ",
        req.uri().to_string(),
        " HTTP/1.1\r\n",
        "Host: ",
        ADDRESS,
        "\r\n\r\n",
    );
    let hash = id_hash(req.uri().to_string());
    if !QUERYMAP.lock().unwrap().contains_key(&hash) {
        if !RESULTMAP.lock().unwrap().contains_key(&hash) {
            QUERYMAP.lock().unwrap().insert(hash, Webitem::new
                (timenow(), httpget.to_string()));
        }
    }
    let hash = id_hash(req.uri().to_string());
    let html_string = redirect_html(hash);
    return html_string.as_bytes().to_vec();
}
```

```
362
363 fn checksum(hash: &str, bytes: &[u8]) -> bool {
364     // Check that hash matches to the hashed bytes
365     hash == &hash512(bytes)[0..hash.len()] // Match the
        begining
366 }
367
368 // We do not trust resources outside of the enclave.
369 // When opening files we check it is not changed.
370 fn style_css() -> &'static [u8] {
371     // We do not trust the unsafe world!
372     let bytes = include_bytes!("../css/styles.css");
373     // Check and verify it is the resource we think it is.
374     assert!(checksum("f2421cb5603eb88797c55d92e979", bytes));
375     return bytes;
376 }
377
378 //*****For verifying Correct Ahmia.fi CSS files are loaded
    *****//
379
380 fn style_arrow() -> &'static [u8] {
381     let bytes = include_bytes!("../css/ddarrow.png");
382     assert!(checksum("3b3f21d6b5644d05e47d1904f39", bytes));
383     return bytes;
384 }
385
386 fn style_ahmiafi() -> &'static [u8] {
387     let bytes = include_bytes!("../css/ahmiafi_black.png");
388     assert!(checksum("308ea08e8e75", bytes));
389     return bytes;
390 }
391
392 fn style_metro() -> &'static [u8] {
393     let bytes = include_bytes!("../css/metro.jpg");
394     assert!(checksum("7653f69165835bde229b0f9c536b2", bytes));
395     return bytes;
396 }
397
398 //*********************************************************//
399
400
```

```
401 //For verifying correct ahmia.fi index page is loaded
402 fn index_page() -> &'static [u8] {
403     let bytes = include_bytes!("../html/index.html");
404     assert!(checksum("a21ff31225bb419b49ec9f792f3d93ef", bytes
        ));
405     return bytes;
406 }
407
408 //Function to run server, load ahmia.fi and return the user
    search results
409 fn run() -> Result<(), Box<dyn std::error::Error>> {
410     // Run server and return query results
411     get_server()
412         .tcp_nodelay(true)
413         .start(move |req| match req.uri().path() {
414             "/search/" => mini_http::Response::builder()
415                 .status(200)
416                 .header("Content-Type", "text/html")
417                 .body(collect_query(&req))
418                 .unwrap(),
419             "/static/images/ddarrow.png" => mini_http::
                Response::builder()
420                 .status(200)
421                 .header("Content-Type", "image/png")
422                 .body(style_arrow().to_vec())
423                 .unwrap(),
424             "/static/images/metro.jpg" => mini_http::Response
                ::builder()
425                 .status(200)
426                 .header("Content-Type", "image/png")
427                 .body(style_metro().to_vec())
428                 .unwrap(),
429             "/static/images/ahmiafi_black.png" => mini_http::
                Response::builder()
430                 .status(200)
431                 .header("Content-Type", "image/png")
432                 .body(style_ahmiafi().to_vec())
433                 .unwrap(),
434             "/static/css/normalize.css" => mini_http::Response
                ::builder()
435                 .status(200)
436                 .header("Content-Type", "text/css")
```

```
437                     .body(style_css().to_vec())
438                     .unwrap(),
439                 "/" => mini_http::Response::builder()
440                     .status(200)
441                     .header("Content-Type", "text/html")
442                     .body(index_page().to_vec())
443                     .unwrap(),
444                 _ => mini_http::Response::builder()
445                     .status(200)
446                     .header("Content-Type", "text/html")
447                     .body(execute_query(&req))
448                     .unwrap(),
449             })?;
450     Ok(())
451 }
452 
453 //****Main Function****//
454 
455 pub fn main() {
456     println!("Privacy Extension for Search Engines (PESE)");
457     println!("Searches results from Ahmia.fi: {}:{}", ADDRESS,
            PORT.lock().unwrap());
458     println!("\nSearch Query Mixer:");
459     println!("Open on WebBrowser: http://127.0.0.1:34455/");
460     if let Err(e) = run() {
461         eprintln!("Error: {:?}", e);
462     }
463 }
464 
465 //**********************//
```

Listing A.1. *PESE Code*