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

This work is licensed under a Creative Commons Attribution International 4.0 License.

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

© 2023 Copyright held by the owner/author(s).

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?

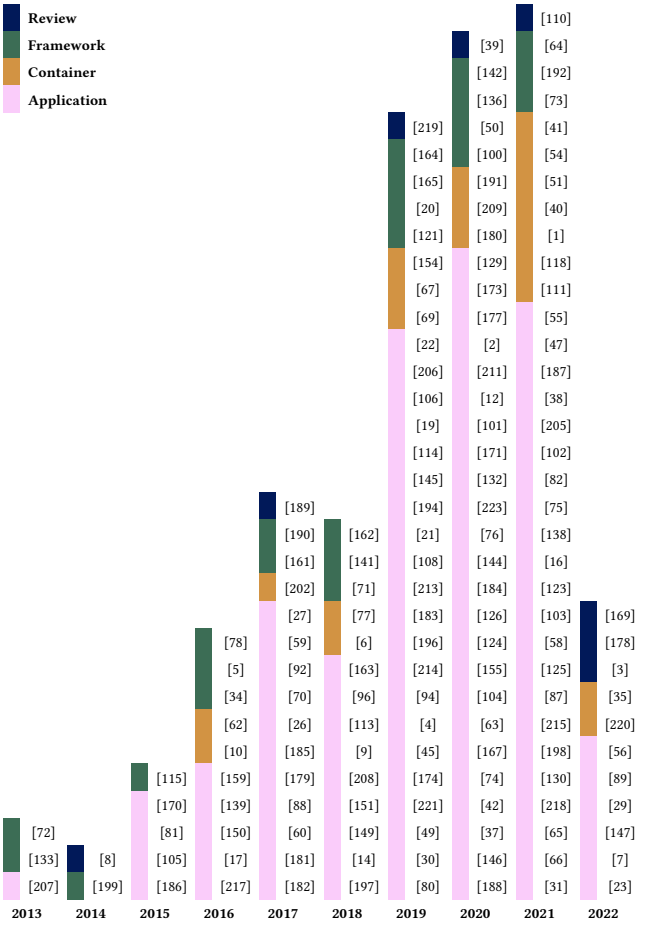

Figure 1: TEE literature falls under the categories of review, framework, container, and application.

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

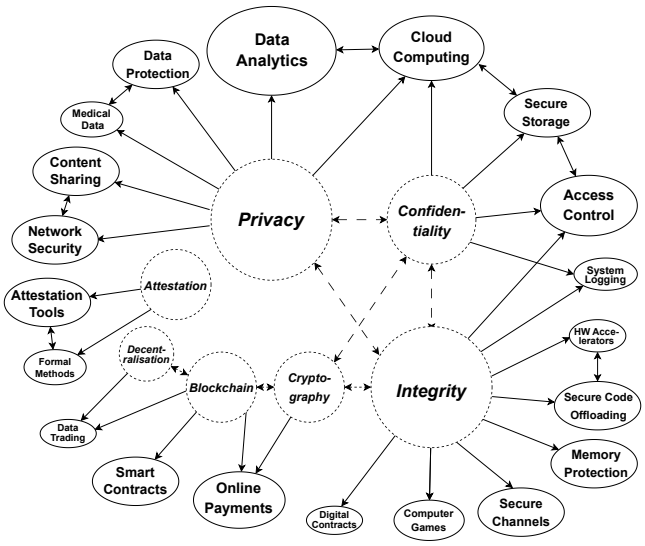

Figure 2: We define which classification aggregates the usage examples after reviewing the TEE example applications. Only the most significant relationships between the categories and the security properties and mechanisms are depicted.

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.

| | | [105] | [81] | [186] | [207] | [170] | [17] | [139] | [36, 159] | [148, 150] | [216, 217] | [185] | [86, 92] | [181] | [60] | [59, 120] | [27] | [182] | [25, 26] | [179] | [88, 204] | [70] | [149] | [113] | [208] | [14, 97] | [18, 197] | [9, 175] | [151, 222] | [96, 158] | [163] | [114] | [194] | [4] | [19, 157] | [80] | [49, 210] | [44, 45] | [30] | [106] | [183] | [206] | [221] | [93, 94] | [195, 196] | [145] | [107, 108, 112] | [160, 213] | [21] | [174] | [22] | [214] | [184] | [146] | [144] | [74] | [126, 127] | [2] | [223] | [101] | [12] | [177, 178] | [104] | [42] | [188] | [173] | [37] | [155] | [63] | [171, 176] | [124] | [76] | [167] | [131, 132] | [211] | [13, 66] | [103] | [187] | [16] | [215] | [55] | [205] | [87] | [102] | [138] | [198] | [130] | [125, 128] | [46, 47] | [122, 123] | [58, 79] | [218] | [64, 65] | [31, 85] | [38] | [75] | [82, 83] | [147] | [28, 29] | [56] | [89] | [7] | [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. secureTF: 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