An Exploration of Trustful, Secure Code Execution and its Applications

Nick Dekker, Arnout van der Knaap, Joas Mulder, Sander Oostmeyer, Justin Segond

***Abstract*—With increasingly many devices in the world, access to them is easier and more plentiful than ever before. This also means that the number of applications and their complexity rises, which brings an undesirable element along; malicious software. To counteract such software, many companies have set out to develop a solution, Trusted Execution Environments (TEE). In this paper, we present a survey of these trusted execution environments. The purpose of this paper is to answer the question: *How can code be executed in a secure and trusted manner?* By exploring existing implementations and the vulnerabilities of TEEs, it is clear that they could benefit from more development. In this paper, security improvements and drawbacks of distributed in comparison to TEEs are given. Future development of new systems based on blockchain technology is found to be a promising alternative to trusted hardware execution. Blockchain technology can provide similar guarantees about the trustful execution of the code but it is currently not very scalable.**

***Index Terms*—Computation, Trust, Security, Blockchain**

I. Introduction

SOFTWARE has become fundamental to our daily lives [1]. As the adoption of software solutions grows in more and more fields, the probability that malicious or faulty code execution could do harm, increases. This is especially true as software solutions are being used in many industries with safety-critical systems, such as medicine, aeronautics and nuclear energy [1]. With the increase in the use of software also comes an increase in scale and complexity of this software. Due to the complexity, code execution is an unpredictable process and prone to a large range of faults. This, in turn, increases the surface of possible malicious attacks. Which emphasizes the need for secure and trusted code execution environments.

The need for a secure and trusted code execution becomes an increasing necessity when systems evolve from a centralized architecture to a decentralized one. In a centralized system, the system has full control about what code is being run. However, in a decentralized system this is not the case as some of the code is being run on another node. If this node is malicious it might not execute the code at all and send the answer that the malicious actors want you to have. There are multiple ways to prevent this kind of behavior, with varying degrees of security, and trust guarantees. Bitcoin [2] and Ethereum [3] have solved the problem of trusted code execution trough global consensus, however, global consensus is a technique that is notoriously slow and involves a lot of network traffic [4]. The time it takes to make a decision is increased to seconds per decision instead of milliseconds and every decision has to be send to all nodes in the network for them to agree or not. For instance, Bitcoin has a theoretically upper bounded transaction throughput of only seven transactions per second [5]. Despite such problems a benefit of decentralized computation can be seen in BOINC [6]. This is a platform were users can volunteer computer resources like processing power to perform resource-intensive scientific workloads. At the time of writing the network was computing 28.232 PetaFLOPS (10^{15} Floating point operations per second) on average over the last 24 hours. It is not economically viable to achieve this kind of computing power in a centralized fashion.

Secure and trusted execution environments are becoming more important in both centralized and decentralized computing. The existing corpus of literature on secure and trusted code execution environments report on their individual techniques and weaknesses [7]–[15]. However, a survey of current trustful and secure code execution techniques, and the application of such techniques is missing. Our study aims to fill this gap in literature, by means of a literature study.

As the goal of the survey is to provide an overview of the available secure and trusted execution environments, we formulate the following research question:
How can code be executed in a secure and trusted manner?
To answer this question we break up this main question into three sub-questions:

1) How can code be trustfully and securely executed? (section III, IV).
2) Which trusted execution platforms have been implemented, and what are their problems? (section V).
3) What are the applications of Trusted execution environments? (section VI).

After answering all the sub questions we will formulate an answer to the main research question in the conclusion (section VII). But before discussing the sub questions, we emphasize on the difference between securely executing code and trustfully executing code. **Security** is the state of being free from danger or threat [16]. In terms of code execution this means that a device and files are free from danger of unjust alteration with malicious intent. **Trustfully** executing code is in many ways almost the same but the point of view is slightly different. Trustfully executing code is about the user sending the code to be run somewhere, whether this is on the same device or somewhere else as in distributed computing, having to trust the code is run correctly and without the device executing the code listening in. A common way to achieve trust is dedicated hardware to run the code on. What constitutes as a trusted execution for the purpose of this paper will be defined in the next section.

II. Trusted Execution

In this paper, the trustful execution of code is discussed. To achieve trustful execution of code, the environment where the code is executed must be protected against interference from outside sources. What constitutes as protected and thus a Trusted Execution Environment (TEE) is not agreed upon in literature or commerce. Therefore, it is useful to state our definition of some of the terms involved in trusted execution.

A. Trusted Applications

Section V-B will present existing hardware implementations, as well as the environments in which they could be executed. These environments and hardware solutions are capable of running applications. At times, these applications are 'regular' ones, without specially designed code. Other times, code has been created with the TEE in mind. These programs are capable of communicating with trusted parts of the hardware. According to Dettenborn, a trusted application is: *An application encapsulating the security-critical functionality to be run within the TEE* [17].

B. Trusted Environments Standardization

Trusted Execution Environment, or TEE for short, is a term that is used often both in literature and by chip vendors. Even though the term is used often, there is no standard definition for TEE. Instead, there are multiple definitions developed by different manufactures and researchers. For a more in depth view of what exactly the differences in opinions and statements about TEE are, we refer the reader to the work of Sabt et al. [18]. For the purposes of this article it is sufficient to say that most definitions agree on isolated execution but secure storage can be involved in varying degrees [18], even though GlobalPlatform has the most clear specifications. Thus in this article isolated execution and secure storage are used as the classification of TEE. This allows for the comparison of implementations with different standards in the paper.

III. Issues in Securely Executing Code

Security and trust are major problems in computer software. Ensuring that a system (and network) is secure and trusted is a difficult task. There are a lot of different ways a system can have its security compromised. With software being more widely used than ever, several problems have risen in the field of security and trust. In this section we will define the levels at which a system should be secure and the issues that have arisen. These issues have to be solved in order to securely execute code.

A. Machine Security

Executing malicious or bugged code can lead to problems on the machine that is executing this code. Machines may contain private or sensitive data. In case the machine is compromised, this data may be stolen. When executing code on a machine, the following should be expected [18]:

1) ***Authenticity****:* The machine should only grant access to important actions when given the correct rights. When the authenticity is not handled correctly, it can lead to compromisation of the system.

2) ***Integrity****:* Machine should not have been tampered with and execute code correctly. In the case code is altered, unexpected events may occur, leading to system errors and potentially harmful situations.

3) ***Confidentiality****:* Code and data should not have been observable by unauthorized applications, or even the main OS of the system. When the confidentiality is not handled correctly, private data or code may be leaked.

B. Issues

Next, the issues that arise when trying to securely and trustfully execute code will be discussed, and how these issues relate to machine security.

1) ***Physical issues****:* In society, security is usually associated with physically securing an object. In the world of secure and trusted computing, physical security is also a problem [19]. A person who has physical access to a machine might be able to steal the data (confidentiality), or take control of the machine by changing the code (authenticity and integrity). This is not a big problem when the machine has no private or sensitive data. When for instance a person has physical access to a central server that stores a lot of users private data, physical access becomes a much bigger problem. The person accessing the machine can possibly steal the data.

2) ***Secure Code Execution****:* The next issue is securely executing code. When executing untrusted code, ensuring that this code can be executed securely and without compromising the machine can be a difficult task. Malicious or bugged code can compromise a machine if not handled correctly. In case of compromise the data on the machine could be stolen, which harms the confidentiality. The machine could also crash. This can lead to loss of data, which harms the integrity of the machine.

3) ***Trusted Code Execution****:* Trusted code execution can also be an issue. When code is sent to another machine in a network to be executed, how is it possible to ensure that this code is executed correctly. Ensuring that these machines give the correct result can be a difficult task. Nodes with malicious intent can give fake results, and therefore subvert the network.

4) ***Code Secrecy Issues****:* The last issue is code secrecy issues. In cloud computing for instance, a client can send code to a cloud computing provider to be executed. When this code is confidential, the client has to trust the cloud computing provider and the employees who have access to the machine to not steal it. Stolen code can be disastrous for the companies that have spent a lot of money developing the code. Because the code is stolen from the machine, this is bad for the confidentiality of the machine.

C. Summary

To summarize, when trying to execute code in a secure and trustful manner there are multiple issues that need to be addressed. Issues not only arise at the machine level, but also

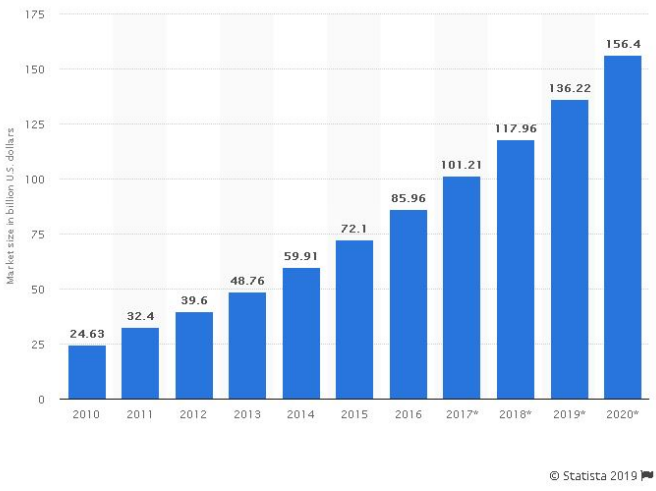

Fig. 1: Total size of the public cloud computing market from 2008 to 2020 (in billion U.S. dollars), image copied from [22]

at the network level. Executing code in a secure and trustful manner involves protecting both the code being run on the machine and protecting the machine from the code being run. The code and its states should not be visible to the machine before, during or after the run, and on top of that, should be safe form someone with physical access to the machine. However, there are techniques available that solve some of the issues addressed. Trusted execution environments are one of the main ways to solve many of these problems. They do this by enforcing trust.

IV. Trust in Remote Code Execution

Trust in remote code execution is becoming increasingly important as computations are outsourced more frequently [20]. An environment which stimulates this development is cloud computing. By first taking a closer look into this technology the increasing relevance of trust in remote code execution is clarified. Furthermore, a definition of what trust is, and how it can be enforced will be given.

A. Cloud Computing

Cloud computing has seen steady growth and adoption in the last ten years, and is projected to keep doing so, see Figure 1. There has been a shift in the way businesses around the world deploy their applications. No longer do these companies require large capital outlays in hardware to deploy their service, or the human expense to operate it [21]. Cloud computing has commoditized computing power, further leveling the playing field between small start-ups and big corporate firms.

Consequently, this rise of cloud computing has led us to an age in which most of our computations and data are handled by a few big cloud computing providers: Amazon, Microsoft and Google. While this does provide businesses with strong economies of scale, it does pose some risks. By centralizing the world's most used services and apps, you are effectively creating a "honeypot" for hackers and other actors with malicious intent [23]. This has resulted in an increasing demand of trust with regards to the use of cloud computing services. The issues and challenges surrounding this have been widely discussed. [23]–[27] In the following sections, a definition of this trust will be given and the ways in which trust is currently enforced and managed.

B. Defining Trust

Cloud computing has commoditized the ability to perform computations externally, removing the need for costly hardware and the corresponding maintenance. However, by using this service, it should provide trust that the execution of the code runs smoothly and gives us the right result. What does this notion of trust mean when it comes to the valid execution of code? For this, we have to take a step back. When running a piece of code, what should be expected from it? A list of requirements was given in section III, for a more in depth explanation of these requirements, we refer the reader to the work of Sabt et al. [18]. When a certain system adheres to these requirements, the respective system can confidently be called trustworthy for executing code.

Currently, several systems exist who enforce these requirements by utilizing several techniques and methods. The following section will examine these techniques and their corresponding efficacy in real world scenarios.

C. Enforcing Trust

As previously discussed, the advent of the internet and cloud computing capabilities, have significantly increased the applicability of executing code remotely. From a security standpoint, this brings several issues. Often, there is not much known about the machine that is running the code, and thus risk the possibility that this machine is malignant, preventing code from being run correctly. This has increased the demand for methods to enforce trust in these situations by leveraging hardware and software solutions. In this section, it will be explored how trust could be enforced.

*1) **Virtualized Environments**:* A key mechanism that is utilized in cloud computing infrastructure is virtualization. Virtualization is defined as the creation of a virtual entity of something, which is usually a virtual instance of an operating system. In cloud computing this can be software, middleware or hardware [28]. Security in cloud computing settings comes down to two objectives: isolating user's workloads so that they cannot affect each other, and making sure the workload of each individual user is secure and authentic. Both objectives relate to our goal of trusted code execution, as is discussed in section II.

Research on cloud security found that existing virtualization methods operate under two assumptions which make these methods unfeasible for securing code execution in a cloud setting [28]. First, the assumption is made that the VMM (Virtual Machine Monitor) has knowledge of the sofware being virtualized, allowing it to do integrity checks and enforce its security this way. Secondly, these methods assume that the Virtual Machine can be monitored from the moment it boots.

This is often not the case in a cloud setting, as guests can easily start up a snapshot, which may be compromised [28].

This means that trusted code execution in a cloud setting can not be achieved by virtualization alone. To this end, a secure version of virtual-machine introspection can be used. This introspection makes no assumptions on the running state of the guest VM and its integrity. The only assumption made is on the hardware, it is presumed an attacker is not able to re-program the CPU. This introspection method further assumes that the hypervisor and security VM are trusted, which allows for the establishment of a dynamic root of integrity. From this root the method is then able to dynamically determine the integrity of all critical components in the VM [28]. This approach was found to have perfect accuracy in detecting malicious behaviour, while only introducing a 2% overhead in macrobenchmarks.

*2) **Dynamic Root of Trust Measurement**:* Building a level of trust when working with unknown systems can be done by utilizing so called roots of trust. These roots of trust are part of the Trusted Computer Base (TCB) of a computer system. The trusted Computer Systems Evaluation Criteria (TCSEC) define the TCB of a computer system as the part of the system (hardware, firmware, software and /or other) that is critical to its security and whose failure (due to bugs or vulnerabilities or any other reason) may lead to compromise of the system [29].

A piece of hardware, called the Trusted Platform Module (TPM), which is attached to the motherboard, is used in many computer systems as a root of trust. It is intended to provide three roots of trust i.e. a) Root of Trust for Measurement (RTM), b) Root of Trust for Storage (RTS), and c) Root of Trust for Reporting (RTR) [30]. RTM is defined as an implementation of a hash algorithm which can put the system in a trusted state. Static RTM (SRTM) is used to maintain a chain of trust for the entire boot chain, which can be very long. Elements in the chain are also subject to change, making it even more challenging. To this end Dynamic RTM (DRTM) was introduced. The DRTM allows the launch of the measured environment at any time without a platform reset, which would normally be necessary to establish a base of trust [30]. DRTM is used in multiple implementations to facilitate trusted code execution [31]–[34].

*3) **Remote attestation**:* Very much related to DRTM is remote attestation. Remote attestation methods define a protocol or line of communication between an authority and a remote machine. The goal of such a protocol is to be able to attest the remote machine for authenticity. The implementations using DRTM are basically remote attestation methods, as the TPM is used to attest the authenticity of the system.

Coker et al. suggests five principles are crucial for attestation architectures [35]. Ideally, these architectures would adhere to all of these principles. But in reality this is often not the case. The following constraints are imposed by these principles, and are thus important in designing effective attestation architectures:

1) Measuring diverse aspects of the target of attestation.
2) Separating domains to ensure measurement tools can prepare their results without interference.
3) The ability to protect itself, at least at a core trust base that can set up this domain separation mechanism.
4) Delegating attestations and sending them to selected peers, allowing the target to select what facts get shared.
5) Managing attestation so that the target is able to enforce certain policies.

Multiple implementations of attestation architectures exist. All of which adhere to at least some of the constraints mentioned above. Most of these involve the need of a hardware component like the TPM to act as the core trust base [36]. Attempts to remove this need for a hardware trust base have been made [37], but have been found to ultimately still require an immense degree of knowledge on the underlying hardware [35].

Lastly, recent advancements in processors have opened the door to some vulnerabilities to existing attestation methods. Modern machines that have multiple processors, introduce virtualization extensions, have a greater variety of side effects, and suffer from nondeterminism [38]. This means modifications to existing attestation methods need to be made in order to be able continue relying on them.

*4) **Trusted Code Execution on Untrusted Hardware**:* All of the techniques for trusted code execution above, operate under the assumption that the hardware of the target system can be trusted (i.e the hardware is not tampered with). When executing code on a remote machine over which we have no physical control, we can not guarantee this machine is using safe hardware. As a result, methods are required which are both resilient to malicious software *and* hardware.

To this end, a Verifiable Computation scheme utilizing Yao's Garbled Circuits with a fully-homomorphic encryption scheme can be used [39]. The efficiency of this protocol is proved to be $O(n+m)$. Besides that, the scheme also provides full privacy of the client's inputs and outputs.

*5) **Blockchains**:* Recently, blockchain technology and the wide ranges of its possible applications have been widely discussed [40]. Proponents of this new technology have been talking about the potential transformative power of distributed ledger technologies for some time. While this technology may provide some beneficial and useful properties [41], its potential in disrupting several market sectors is likely overestimated [42].

However, some of these properties do give us useful primitives to design systems in which code can be executed trustfully, namely:

1) *Public Verifiability*: Allows anyone to verify the correctness of the state the system is in.
2) *Integrity*: Ensures that information is protected from unauthorized modifications, i.e. that retrieved data is correct.
3) *Transparency*: Related to public verifiability, transparency allows all participants in the network to verify the state of the network effectively.

There are several implementations using blockchain technology, that allow the secure execution of code, one of which is Ethereum [43]. Permissionless blockchain implementations (i.e blockchains in which any user can take part), need some

	PoW	PoS	PoET	BFT and variants	Federated BFT
Blockchain type	Permissionless	Both	Both	Permissioned	Permissionless
Transaction finality	Probabilistic	Probabilistic	Probabilistic	Immediate	Immediate
Transaction rate	Low	High	Medium	High	High
Token needed?	Yes	Yes	No	No	No
Cost of participation	Yes	Yes	No	No	No
Scalability of peer network	High	High	High	Low	High
Trust model	Untrusted	Untrusted	Untrusted	Semi-trusted	Semi-trusted
Adversary Tolerance	<=25%	Depends on specific algorithm used	Unknown	<=33%	<=33%

Fig. 2: A comparison of popular blockchain consensus mechanisms, image copied from [45]

form of a consensus protocol to enforce trust. Ethereum, as of right now, uses a consensus protocol called ”Proof of Work”. This protocol relies on a network of ”miners” to ensure the validity of the network. These miners are ultimately the remote machines responsible for executing the code of the users. Proof of work was initially invented by Bitcoin [2], and is proven to be quite resilient to various kinds of attacks [44]. Lastly, networks like Ethereum allow anyone to start mining, which means blockchain implementations like Ethereum provide trusted code execution under no assumptions on the hardware or software of these miners.

Other forms of consensus protocols exist as well, each having their own benefits and drawbacks. A list of these protocols is given below [45], as well as a comparative table (Figure 2).

1) *Proof of Stake (PoS)*: PoS completely replaces the mining operation with an alternative approach involving a users stake or ownership of virtual currency in the blockchain system. Instead of needing to buy hardware to participate in these kind of blockchain systems, one would need to buy the underlying cryptocurrency to increase their odds of becoming a validator.
2) *Proof of Elapsed Time (PoET)*: PoET utilizes a TEE, powered by Intel SGX [46] to create new blocks in a trustless manner. SGX is discussed in more detail in section V V-B. Each round a random validator is selected using a lottery based election model. Using a TEE guarantees the safety and randomness of this process.
3) *Practical Byzantine Fault Tolerance (PBFT)*: PBFT is a state machine replication algorithm that tolerates Byzantine faults provided fewer than a third of the replicas are faulty. BFT provides linearizability, which is a strong safety property, without relying on any synchrony assumption [47].
4) *Federated Byzantine Agreement*: Consensus models that are a derived form of Byzantine Fault tolerance algorithms modified to include open-ended participation from users [45].

D. Summary

To summarize, many methods exist to enforce trusted execution of code. This enforcement often relies on the assumption that the hardware in a system can be trusted and used as a root of trust. In decentralized systems, this is not necessary, at the cost of additional overhead and redundancy. This leads to a system with lower efficiency when executing code. To trustfully and securely execute code, the environment in which the code is executed needs to be controlled. This controlled environment can be achieved by using special hardware as a base of trust. Another way controlling can be done is with overhead in a decentralized system. An overview of the specific implementations of trusted environments and there problems will be provided in the next section.

V. TEE Implementations and Problems

In this section, implementations of systems that enforce trust will be discussed. As explained in Section II, there are different ideas of what exactly trust entails. In this section, a list of hardware solutions and software environments will be provided that enforce some form of trust. There is no perfect solution for trusted execution yet, trade-offs exist for each implementation. To understand what trade-offs are made, we give an overview of trusted code execution and their problems.

A. Overview of Types

There are several ways to implement trusted code execution. Before diving into these, it is important to go into a little more detail about Trusted Execution. As explored in section III, the term trusted code execution can be interpreted in multiple ways [18], [48]. First, when looking from a confidentiality perspective, one might argue that the code - and its data set - being run should be readable only by those with the correct privileges. However, if the code is open-sourced, integrity is the main focus, with correct code execution being paramount. Lastly, authenticity is an important aspect as well, promising of the safety of a users system. These three points warrant different measures.

First, we will touch on *confidentiality*. Often used for maintaining confidentiality is a hardware solution, where a CPU is protected by hardware encryption. These CPUs often have a separate partition on which the ’trusted code’ is executed. Most of the time these partitions serve a specific purpose concerning sensitive data. An example of this can be found in banking applications [49]. Next, *integrity*, which is a way of making sure code is executed properly is by executing code on multiple machines. Either by re-running the same operations multiple times (and open-sourced), or running parts of the code on different machines. The latter of which may protect the data if code is confidential, more on that later (V-B). Additionally, code can be self-verifying [7], meaning, the code is protected against modification. Such code might be asserted

or validated in several ways. A well-known type of self-verifying code is (cryptographic) checksumming. Another way of code validation is watermarking, this allows the computing device to check for the presence of the intact watermark [7]. As described in [7], these methods are not sufficient, as they are solely protected by software. In [50] is proposed that hardware is necessary to stop software duplication (and therefore modification). Lastly, *authenticity* means the user is who they say they are. No one should be able to access parts they are not authorized to. In TEEs, this problem is solved by not giving access to the secure area. Instead, data or actions are requested and the secure environment either allows or disallows the request.

B. Overview of Existing Implementations

As mentioned before, there are hardware solutions preventing code execution from failure or interception. Since 1997 [9] there have been designs and ideas about trusted environments. In this section, we take a look at some implementations and identify any problems and shortcomings they posses.

Intel SGX - First off, Intel SGX (Software Guard Extension) is a prominent contender in the trusted computing space at this time. The processor is separated into several sections, the main OS runs on one part, while trusted applications run on the TEE. Segments inside that TEE running code are called enclaves. Despite its careful design there are doubts about its security [10]. So-called side-channel attacks can still exploit vulnerabilities [46]. Using these attacks, sensitive data is still subject to interception. According to [46], these attack threats are eliminated by disabling hyper-threading. Despite this vulnerability, SGX is safer than regular chips. A security effectiveness analysis of SGX [11] shows that the code inside SGX enclaves can be identified. Schwarz [51] has recently experimented with running malicious code from inside SGX. This experiment resulted with the malicious code in full control of the secure enclave as it could substitute the encapsulating software without being detected. Therefore it is not believed that SGX has reached maturity to the point of adoption and trust:

> *We conclude that instead of protecting users from harm, SGX currently poses a security threat, facilitating so-called super-malware with ready-to-hit exploits.*

ARM TrustZone - Mobile devices could also profit from additional security. ARM TrustZone [12] provides such functionality for mobile devices. Like Intel SGX, TrustZone isolates two segments in memory where code is executed. Though, unlike SGX, TrustZone runs a fully isolated OS. By running a ”secure world” on top of a secondary OS (Figure 3) the trusted applications are isolated from regular applications.

TrustZone does however have some vulnerabilities inhibiting its efficacy in being a TEE [52]. A vulnerability exists in the monitor, whose exploitation allows the execution of code in the most privileged exception level of the CPU. Qualcomm’s TrustZone implementation further allows the operating system to load in binaries to expand the functionality of the execution environment. These binaries, called trustlets, allow an attacker to execute an arbitrary piece of code in the Secure World, see Figure 3.

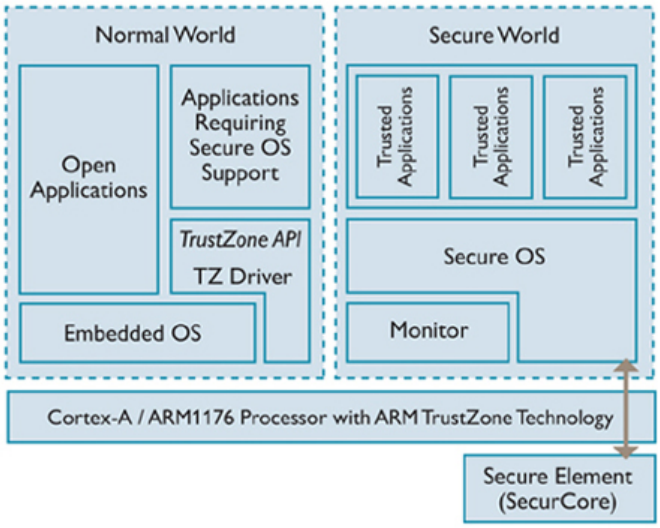

Fig. 3: The compartmentalization of ARM TrustZone, image copied from [53]

AMD SEV - AMD SEV is another solution that focuses heavily on trusted code execution. In contrast to Intel SGX and ARM TrustZone, this processor aims to isolate sections on a virtual machine specifically. With this separation, entirely secure key generating is made possible. Since a key is used to encrypt system memory, created at boot, AMD SEV is usable with any operating system [13].

Mobile processor manufacturers also have shown interest in offering (hardware) security. This mostly means sealing off storage using encryption. Sensitive activities, like banking would also become more secure.

Unfortunately, a vulnerability was found that is able to reliably and efficiently extract all memory contents from SEV-encrypted virtual machines in plain text [14]. Morbitzer et al. state that their implementation, called SEVered, is based on the observation that the page-wise encryption of main memory lacks integrity protection. They propose that the best solution would be to provide a full-featured integrity and freshness protection of guest-pages additional to the encryption, as realized in Intel SGX.

Another study done by Mofrad et al. [54] states the same vulnerabilities in relation to memory integrity protection. Furthermore, they state SEV is vulnerable to DOS attacks and memory side-channel attacks. According to Mofrad et al. SEV provides weaker security protections than Intel SGX.

Keystone - Although Keystone [55] is currently unfinished, it is still worth mentioning here as this is the only fully open-source project to build a trustworthy secure hardware enclave. All other trusted hardware solutions rely partially on security by obscurity as parts of the hardware-stack implementation are not made public. This project aims to change that and pursues open security.

Sanctum - An open source design of a secure processor [10]. It consists of a small hardware change to processors and security monitor software. Sanctum aims to protect running

software against a range of software attacks. It offers the same software security promises as Intel SGX, with the added security of preventing software side channel attacks. At the hardware level, the system's memory is divided into regions, which use disjoint cache sets [56]. Each of these regions can only be allocated to one enclave. There is no leakage of private memory access patterns through the cache, because the processor's memory is divided into these regions. Unfortunately, Sanctum offers no protection against hardware based attacks.

Aegis - Aegis [15] was one of the first implementations of hardware trusted execution environments when it was released in 2003. The architecture provides protection against physical attacks and several software attacks. Aegis provides two security levels for the applications. The first level protects the integrity of the executing software. The second level protects the confidentiality and the authenticity as well. Aegis does not offer protection against software side channel attacks [57]. There is also no support for vitalization, which is key in the more recent implementations of trusted execution environments.

Secure Element - Besides fully integrated trusted hardware elements, there are Secure Elements (SE). They are separate, small chips, used in NFC cards to provide secure data exchange. The chip was heavily used in GSM phones, prior to smartphones. Nowadays, these chips are still in use to verify transactions [58], providing security tokens to ensure authenticity.

All of the implementations mentioned above are usable to achieve trusted computing. They are used in systems to create environments that allow users to shield themselves from attacks.

C. Environments Enforcing Trusted Computing

New software methods have been developed for trusted computing. Some of these are an Operating System (OS) that run on the hardware discussed in V-B. Other environments are software systems that are created to ensure secure code execution. The degree of trust enforced differs between different solutions. Discussed are some of the solutions currently available.

Andix OS - Devised as a result of the notion of malicious applications running in the same memory space as programs utilizing sensitive information [59]. The OS calls itself ARM TrustZone aware, supporting this secure hardware and using it to create and run trusted applications. Andix targets developers that desire to create trusted applications. Andix OS provides guarantees about trust separating trusted and untrusted applications and not letting them access the same memory. If an untrusted program tries to access memory that is trusted the program will be stopped from doing so by this OS.

Genode TEE - Like Andix, Genode makes use of ARM TrustZone. Genode runs in the secure world of ARM TZ, while another (Linux) OS runs in the normal world. Genode allows the GPU to be used, while still showing that code execution, visible on screen is taking place in the normal world.

BOINC - BOINC is a software system designed for public resource computing [6]. In other words, a system where calculations can be executed on certain problems in the BOINC network by participants. As explained in [6], BOINC works with a master URL, which is at the same time the home page of the website. Members can participate by registering projects to this homepage. However public resource computing is faced with the problem that calculation can be wrong, possible reasons for this could be malfunctioning computers or mischievous members. The way BOINC enforces trusted result is by implementing redundant computing. Redundant computing is a system where each project can choose a number of replications. If the results to a computation are identical for all replicas, a consensus is reached about the answer of the equation. If no consensus is a found a new result set is created and the process repeats itself.

Blockchain - Blockchain technology is a potential TEE as well. It allows programs to run on its decentralized nodes. These programs are called Smart Contracts [60]. While they are bounded in computation time, they offer an interesting use. Due to programs being uploaded and run in a decentralized environment, its validity is verified without the need for trusted hardware, making it an excellent candidate for trusted computing.

We will look at Ethereum and its Virtual Machine (EVM) [43] as a potential TEE. While the EVM is not defined as a TEE, we research the possibility of executing code on a decentralized environment and its effect on the degree of trust. When a program is uploaded to a public chain, such as the EVM, its legitimacy can be verified by anyone. The advantages, disadvantages and possibilities of uses and some implementations of TEEs are discussed in VI.

Some blockchains allow for secure implementations on their programmable environment. Trustee [61] is a Vickrey auction implementation (sealed bid auction where the highest bidder wins, but pays the amount of the second highest bidder) which preserves privacy. Trustee runs on Ethereum, using Intel SGX. Secure computations are executed to ensure tamper-free results. Trustee is an example of a system based on blockchain that also has confidentiality as a property, but does this by making use of secure hardware.

D. Confidentiality in Trusted Environments

Confidentiality can be achieved using secure hardware, but there are also other methods that can be used. Using some of these methods in combination with a distributed system like EVM can allow for confidential computations that are both verifiable on the blockchain and confidential. Listed are techniques that enforce confidentiality that see use and are relevant in current technology.

1) Homomorphic encryption: a method of processing data while the data is encrypted. The idea of homomorphic encryption is that encrypted functions executed on encrypted data will result in the same as a function on the plaintext data.
2) secure Multi-Party Computation (sMPC): the act of executing code on multiple machines, breaking up the data sets (and functions) in such a way the nodes are not aware of what they are computing.

implementation	specifications	hardware attack protection	side-channel attack protection	attack surface
Intel SGX	Proprietary	yes	yes*	Small
ARM TrustZone	Global Platform	yes	yes	Medium
AMD SEV	SEV ES	yes	yes*	Medium
Sanctum	N/A	no	yes	Small
Aegis	N/A	yes	no	Large

∗ has vulnerabilities

TABLE I: A comparison of hardware implementations [14], [51], [52], [56], [57], [62]–[64]

3) zero knowledge proof (ZKP): using this technique, a node is able to prove to another node it knows a fact or value, without revealing said value.

Using one, or multiple of these methods, obfuscation of the code is applied. Although these techniques might be effective, they are not optimal as overhead is carried along, albeit small overhead with homomorphic encryption [65]. An example of a system that uses some of these techniques is Enigma [66]. Enigma is based on a blockchain, moreover, it uses sMPC for its computations and ZKP to verify computation results. This allows for confidential code execution without trusted hardware.

Another way that privacy can be achieved on a blockchain is trough anonymity [67]. This anonymity might be hard to achieve for some users as analysis of the transactions could still potentially link a user to their node on the chain.

E. Summary

To answer the question:"*Which trusted execution platforms have been implemented, and what are their problems?*" the following topics were discussed. First, an overview of the types of issues in trust and a practical example for each was given. Then an overview of the existing hardware implementations was provided. Table I gives a high level overview of the hardware TEE implementations that have been discussed in this section. It should be noted that the attacks service metric, where small, medium and large is used, was chosen based on the attack surface relative to each other. *Small* means the attack surface is nearly optimal. There are almost no vulnerability points to exploit. *Medium* implies a greater amount of points were found. Then there is *large*, where the problem is more severe and many points are exposed for exploitation. Following the hardware implementation, the environments were discussed that provide trusted execution. Some of these environments rely on trusted hardware like SGX but some of these do not. Finally, ways to enforce confidentiality in distributed systems were discussed. All of the hardware implementations and environments have some vulnerability. Depending on the use case, different systems are appropriate.

VI. Applications of TEEs

The fact that there are so many TEEs is no mere coincidence. Many areas of computing require security and trust. This section lays out some important uses of the TEE solutions, although we do not stop there. We explore the possibilities of running code in a trusted manner on decentralized systems. We briefly explain the idea of decentralized computing, before presenting an overview of advantages and disadvantages of applications of trusted computing.

A. Decentralized Computing

In the last decade decentralized computing has become drastically more popular [68]. The introduction of blockchain technology has played a big part in this development, as it has proven to be more useful in certain applications than "traditional" centralized computing.

In order to provide a better view of decentralized computing, it should first be explained how centralized computing works. In simple terms, centralized computing is a type of architecture where there is one central server that may be connected with one or several clients. This central server is the most important part in centralized computing as it runs the main application and is responsible for the communication between clients who may be connected to the network. For these reasons the main server usually has huge computing power and high data storage at its disposal so it can successfully do its job.

However, as Gray [69] states in his paper, in decentralized computing, there is no central server which runs the main application, but rather the network of clients that runs the applications. The clients are responsible for the calculations, data storage and communication. The way these clients communicate with each other is by a message protocol, where each client can send a message to any client in the network. However, as a consequence, a decentralized architecture does raise some problems. In a decentralized system there is no true global consensus on the state of the network. messaging efficiency is also something these systems struggle with, as all clients have to communicate with each other to make decisions. Potential advantages, however, are the amount of processing power available in large decentralized systems and the added privacy as not all data is collected in one place.

B. Real-world Uses of TEEs

Banking - One use for trusted execution environments is banking [70]. If a banking application is not secure or running on untrusted hardware there is a potential security breach and great sums of money might be lost or private exchanges exposed.

Cryptocurrency - Much related to banking are cryptocurrencies like Bitcoin. Bitcoin, and other cryptocurrencies, have to be sure which transactions on its network are authentic. In contrast to traditional, centralized banking systems, cryptocurrencies are platforms which do not use trusted hardware and cannot rely on it. There is a different level of trust; transactions are public and verified by arbitrary nodes [2].

Instead of having to trust a centralized server, the actions done by an individual are trusted, but that may be unjustified. This problem of trust by using a public ledger. All nodes in the network can exchange currency with each other. As only verified transactions make it into blocks on the chain, the only way to steal from someone is by having more than 50% computing power of the network, which is difficult to achieve [71]. Furthermore, such actions are likely to be detected within minutes if not seconds as it requires large changes to the chain [72]. Which will then lead to the value of the coin dropping, as the trust in the coin drops, making the theft of the currency not economically viable. This still leaves the problem of trustfully exchanging different cryptocurrencies however. Exchanging cryptocurrencies relies either on a trusted third party or on trusting the other person. Trusting the other person leaves the obvious security deficit of one party sending the value to be exchanged first, at which point the other party could leave. A solution for these kinds of situations is provided by Tesseract. Tesseract is real-time cryptocurrency exchange based on trusted hardware [73]. This system relies on Intel SGX enabled servers to create an atomic exchanges between cryptocurrencies possible.

Smart Cards - Secure Element (V-B) chips are often used in cards for authentication and transaction verifications, also known as Smart Cards. They are used in many areas, where wireless transactions are useful [74]–[77]. Some alternatives are suggested in [78], Universal Integrated Circuit Card (UICC) being the most proficient one.

Learning - Another application of secure hardware solutions is in the training of an self-learning algorithm on classified data. If multiple companies want to train an algorithm on a combination of their respective data, while not wanting to share this data with each other, a trusted hardware solution for the training could provide an answer. The trusted hardware just takes inputs from multiple sources and trains the algorithm. The data is never shared with the other party and thus remains confidential.

Biometrics - Recent technological advancements in smartphone technology have allowed for the use of biometrics for fast identification/authentication [79]. In technical terms, biometrics is the automated technique of measuring a physical characteristic or personal trait of an individual and comparing that characteristic or trait to a database for purposes of recognizing that individual [80]. These traits and characteristics are very personal. Thus, several privacy concerns have been raised [81]. TEEs could be utilized to ensure these privacy concerns are never violated. For example, one of the concerns involves the biometric information being ”captured” in the primary market. When this has occurred, the information can easily be replicated, copied, and otherwise shared among countless public- and private-sector databases [82]. TEEs offer us a solution in which we can ensure that this biometric information is not captured by any external actors.

C. Trusted Code on Distributed Systems

Systems have been implemented that offer a trusted environment without using trusted hardware. An interesting implementation for distributed computing is Enigma. Enigma is a platform that provides trustful execution of code using a blockchain as a base and a shared hashtable for data sharing [66]. Enigma is faster than other blockchain solutions because not every node checks the result of computations. Instead, a zero knowledge proof is used to prove computations are run. This means that the scaling factor, that is such an issue for most blockchains, is less of a factor. Enigma also uses sMPC to improve the privacy of the code execution. Thus, although no trusted hardware is used, it still provides a trusted environment for code execution. As can be seen in an implementation such as Enigma, smart contracts, running on a blockchain, are usable as a TEE. It solves the issue of trust intrinsically with its consensus protocol. All the while, isolation of sensitive information is preserved using obfuscation and encryption. We make a case for these distributed systems replacing, or at least providing an alternative for hardware TEEs. As shown, distributed computing can be used to create a TEE on on trusted hardware, there is one major downside. They have significant overhead as computations need to be verified multiple times to create the required trust. TEEs based on hardware implementations do not have this overhead but they also have vulnerabilities as is shown in Table I.

D. Summary

In all, it seems several useful applications of TEEs exist. These applications range widely, from banking to machine learning to biometrics. The applications indicate the growing necessity for TEEs in several scenarios. Hardware-based TEEs can not always be implemented, as there is no control over all participants in the network. In these cases, decentralized blockchain solutions can be implemented to offer secure and trusted code execution.

VII. Conclusion

Finally, to answer the question: *“How can code be executed in a secure and trusted manner?”*, a brief summary of the conclusions to the sub-questions will be given before drawing a conclusion.

Executing code **securely** requires isolation and robust systems. Malicious or buggy code should not be able to alter or damage the core components of a system.

Subsequently, executing code **trustfully** comes down to three principles, those being authenticity, integrity and confidentiality. We found that *VMMs*, a *DRTM* and *remote attestation* can all be utilized to enforce these requirements. Unfortunately, we also found that recent technological developments in processors have undermined the effectiveness of some of these methods.

TEEs aim to offer us both secure and trusted execution, however, their designs and implementation vary widely. Intel SGX, a prominent contender in the TEE space, shows some promise. However, several vulnerabilities exist, indicating that it is not yet ready for adoption. Furthermore, implementations such as Intel SGX and AMD SEV rely on specialized hardware, restricting the amount of possible use cases. Projects

like Keystone Enclave aim to create the next generation of TEEs without assumptions on the hardware.

Other, more distributed systems, might also be used to execute code in a secure and trusted manner. BOINC, a system which enforces trust in computing through redundancy, has proven quite effective and useful. Similar to this, blockchains have shown to be capable of serving as a TEE. Through several kinds of consensus protocols, trusted code execution in a decentralized fashion is achieved. However, scalability in these systems remains an issue for now as considerable overhead is required to achieve trust.

Several ways of executing code in a secure and trusted manner have been examined in this paper. Some more effective than others. Hardware implementations of TEEs (e.g. SGX, SEV) provide us with better speed and efficiency, but at the cost of requiring specific hardware. Distributed and decentralized solutions often do not operate under any hardware assumptions, but at the cost of speed and efficiency. Both options are useful depending on the use case. Recent advancements in combating the scalability issue in these systems seem promising. Blockchain technology is still in its early stages, and much remains to be researched.

REFERENCES

[1] W. E. Wong, R. Gao, Y. Li, R. Abreu and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[2] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf.

[3] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *White paper*, 2014.

[4] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International workshop on open problems in network security*, Springer, 2015, pp. 112–125.

[5] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gfffdfffdn Sirer, D. Song and R. Wattenhofer, "On scaling decentralized blockchains," vol. 9604, Feb. 2016, pp. 106–125, ISBN: 978-3-662-53356-7. DOI: 10.1007/978-3-662-53357-4_8.

[6] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, IEEE Computer Society, 2004, pp. 4–10.

[7] A. Maña and A. Muñoz, "Trusted code execution in javacard," in *International Conference on Trust, Privacy and Security in Digital Business*, Springer, 2007, pp. 269–279.

[8] J. Criswell, A. Lenharth, D. Dhurjati and V. Adve, "Secure virtual architecture: A safe execution environment for commodity operating systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 351–366, 2007.

[9] W. A. Arbaugh, D. J. Farber and J. M. Smith, "A secure and reliable bootstrap architecture," 1996.

[10] V. Costan, I. Lebedev and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 857–874.

[11] D. Kim, D. Jang, M. Park, Y. Jeong, J. Kim, S. Choi and B. B. Kang, "Sgx-lego: Fine-grained sgx controlled-channel attack and its countermeasure," *Computers & Security*, vol. 82, pp. 118–139, 2019.

[12] A. ARM, "Security technology building a secure system using trustzone technology (white paper)," *ARM Limited*, 2009.

[13] D. Kaplan, J. Powell and T. Woller, "Amd memory encryption," *White paper*, 2016.

[14] M. Morbitzer, M. Huber, J. Horsch and S. Wessel, "Severed: Subverting amd's virtual machine encryption," in *Proceedings of the 11th European Workshop on Systems Security*, ACM, 2018, p. 1.

[15] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk and S. Devadas, "Aegis: Architecture for tamper-evident and tamper-resistant processing," ACM Press, 2003, pp. 160–171.

[16] *Security*, https://en.oxforddictionaries.com/definition/security, Accessed: 2019-03-22.

[17] T. Dettenborn, "Open virtual trusted execution environment," 2016.

[18] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 1, 2015, pp. 57–64.

[19] D. Hutter, "Physical security and why it is important," *Swansea: SANS Institute*, 2016.

[20] R. Buyya, C. S. Yeo and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *2008 10th IEEE international conference on high performance computing and communications*, Ieee, 2008, pp. 5–13.

[21] A. D. JoSEP, R. KAtz, A. KonWinSKi, L. Gunho, D. PAttERSon and A. RABKin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010.

[22] *Total size of the public cloud computing market from 2008 to 2020 (in billion u.s. dollars)*, https://www.statista.com/statistics/510350/worldwide-public-cloud-computing/, Accessed: 2019-03-18.

[23] A. Ghosh and I. Arce, "Guest editors' introduction: In cloud computing we trust-but should we?" *IEEE security & privacy*, vol. 8, no. 6, pp. 14–16, 2010.

[24] S. M. Habib, S. Hauke, S. Ries and M. Mühlhäuser, "Trust as a facilitator in cloud computing: A survey," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 1, no. 1, p. 19, 2012.

[25] C. Everett, "Cloud computing–a question of trust," *Computer Fraud & Security*, vol. 2009, no. 6, pp. 5–7, 2009.

[26] B. Michael, "In clouds shall we trust?" *IEEE Security & Privacy*, vol. 7, no. 5, pp. 3–3, 2009.

[27] A. Coveillo, H. Elias, P. Gelsinger and R. Mcaniff, "Proof, not promises: Creating the trusted cloud," *RSA White Paper, http://www. rsa. com/innovation/docs/11319_TVISION_WP_0211. pdf*, 2011.

[28] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra and D. Zamboni, "Cloud security is not (just) virtualization security: A short paper," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ACM, 2009, pp. 97–102.

[29] D. C. Latham, "Department of defense trusted computer system evaluation criteria," *Department of Defense*, 1986.

[30] S. Romana, H. Pareek and L. Eswari P R, "Dynamic root of trust and challenges," *International Journal of Security, Privacy and Trust Management*, vol. 5, pp. 01–06, May 2016. DOI: 10.5121/ijsptm.2016.5201.

[31] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter and A. Seshadri, "Minimal tcb code execution," in *2007 IEEE Symposium on Security and Privacy (SP'07)*, IEEE, 2007, pp. 267–272.

[32] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter and H. Isozaki, "Flicker: An execution infrastructure for tcb minimization," in *ACM SIGOPS Operating Systems Review*, ACM, vol. 42, 2008, pp. 315–328.

[33] *Trusted boot (tboot)*. [Online]. Available: https://sourceforge.net/projects/tboot/.

[34] K. Eldefrawy, G. Tsudik, A. Francillon and D. Perito, "Smart: Secure and minimal architecture for (establishing dynamic) root of trust.," in *NDSS*, vol. 12, 2012, pp. 1–15.

[35] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy and B. Sniffen, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, no. 2, pp. 63–81, 2011.

[36] L. Gu, X. Ding, R. H. Deng, B. Xie and H. Mei, "Remote attestation on program execution," in *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, ACM, 2008, pp. 11–20.

[37] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. Van Doorn and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *ACM SIGOPS Operating Systems Review*, ACM, vol. 39, 2005, pp. 1–16.

[38] M. Kiperberg, A. Resh and N. J. Zaidenberg, "Remote attestation of software and execution-environment in modern machines," in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, IEEE, 2015, pp. 335–341.

[39] B. J. Parno, "Trust extension as a mechanism for secure code execution on commodity computers," 2010.

[40] S. McLean and S. Deane-Johns, "Demystifying blockchain and distributed ledger technology–hype or hero," *Computer Law Review International*, vol. 17, no. 4, pp. 97–102, 2016.

[41] K. Wüst and A. Gervais, "Do you need a blockchain?" In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, IEEE, 2018, pp. 45–54.

[42] M. Pisa and M. Juden, "Blockchain and economic development: Hype vs. reality," *Center for Global Development Policy Paper*, vol. 107, p. 150, 2017.

[43] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[44] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, ACM, 2016, pp. 3–16.

[45] A. Baliga, "Understanding blockchain consensus models," in *Persistent*, 2017.

[46] V. Costan and S. Devadas, "Intel sgx explained.," *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[47] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[48] J. Huang and D. M. Nicol, "Trust mechanisms for cloud computing," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, p. 9, 2013.

[49] K. Kostiainen, J.-E. Ekberg, N. Asokan and A. Rantala, "On-board credentials with open provisioning," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ACM, 2009, pp. 104–115.

[50] O. Goldreich, "Towards a theory of software protection," in *Conference on the Theory and Application of Cryptographic Techniques*, Springer, 1986, pp. 426–439.

[51] M. Schwarz, S. Weiser and D. Gruss, "Practical enclave malware with intel sgx," *ArXiv preprint arXiv:1902.03256*, 2019.

[52] J. Guilbon, *Trustzone attack surface*, 2018. [Online]. Available: https://blog.quarkslab.com/attacking-the-arms-trustzone.html.

[53] *The compartmentalization of arm trustzone*, https://blog.quarkslab.com/resources/2018-06-18-trustzone/images/TrustZone.jpg, Accessed: 2019-03-19.

[54] S. Mofrad, F. Zhang, S. Lu and W. Shi, "A comparison study of intel sgx and amd memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, ACM, 2018, p. 9.

[55] *Keystone*, https://keystone-enclave.org/, Accessed: 2019-04-04.

[56] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas and S. A. Seshia, "A formal foundation for secure remote execution of enclaves," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: ACM, 2017, pp. 2435–2450, ISBN: 978-1-4503-4946-8. DOI:

10.1145/3133956.3134098. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134098.

[57] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk and S. Devadas, "Aegis: Architecture for tamper-evident and tamper-resistant processing author retrospective," 2015.

[58] M. Langheinrich, "A survey of rfid privacy approaches," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 413–421, 2009.

[59] A. Fitzek, *Development of an arm trustzone aware operating system andix os*, 2014.

[60] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[61] H. S. Galal and A. M. Youssef, "Trustee: Full privacy preserving vickrey auction on top of ethereum,"

[62] M. Papermaster. (2018). Initial AMD Technical Assessment of CTS Labs Research, [Online]. Available: https://community.amd.com/community/amd-corporate/blog/2018/03/21/initial-amd-technical-assessment-of-cts-labs-research (visited on 10/04/2019).

[63] J. Götzfried, M. Eckert, S. Schinzel and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*, ACM, 2017, p. 2.

[64] D. Kaplan, "Protecting vm register state with sev-es," *White paper, Feb*, 2017.

[65] C. Gentry, S. Halevi and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2012, pp. 465–482.

[66] G. Zyskind, O. Nathan and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," *ArXiv preprint arXiv:1506.03471*, 2015.

[67] M. Luongo and C. Pon, "The keep network: A privacy layer for public blockchains," Tech. rep. URl: https://keep. network/whitepaper, Tech. Rep.

[68] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman *et al.*, "Blockchain technology: Beyond bitcoin," *Applied Innovation*, vol. 2, no. 6-10, p. 71, 2016.

[69] J. N. Gray, "An approach to decentralized computer systems," *IEEE Transactions on Software Engineering*, no. 6, pp. 684–692, 1986.

[70] F. Mennes. (2018). PSD2: Creating a Secure Execution Environment for Mobile Banking Apps, [Online]. Available: https://www.onespan.com/blog/psd2-secure-execution-environment-for-mobile-banking-apps (visited on 11/04/2019).

[71] D. Bradbury, "The problem with bitcoin," *Computer Fraud & Security*, vol. 2013, no. 11, pp. 5–8, 2013.

[72] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu and J. Kishigami, "Blockchain contract: Securing a blockchain applied to smart contracts," in *2016 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, 2016, pp. 467–468.

[73] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware.," *IACR Cryptology ePrint Archive*, vol. 2017, p. 1153, 2017.

[74] X. Leroy, "Bytecode verification on java smart cards," *Software: Practice and Experience*, vol. 32, no. 4, pp. 319–340, 2002.

[75] B. A. Aubert and G. Hamel, "Adoption of smart cards in the medical sector:: The canadian experience," *Social Science & Medicine*, vol. 53, no. 7, pp. 879–894, 2001.

[76] G. Kardas and E. T. Tunali, "Design and implementation of a smart card based healthcare information system," *Computer methods and programs in biomedicine*, vol. 81, no. 1, pp. 66–78, 2006.

[77] M.-P. Pelletier, M. Trépanier and C. Morency, "Smart card data use in public transit: A literature review," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 4, pp. 557–568, 2011.

[78] M. Reveilhac and M. Pasquet, "Promising secure element alternatives for nfc technology," in *2009 First International Workshop on Near Field Communication*, IEEE, 2009, pp. 75–80.

[79] P. Sharma. (2017). More Than One Billion Smartphones with Fingerprint Sensors Will Be Shipped In 2018, [Online]. Available: https://www.counterpointresearch.com / more - than - one - billion - smartphones - with - fingerprint - sensors - will - be - shipped - in - 2018/ (visited on 11/04/2019).

[80] B. Miller, "Everything you need to know about automated biometric identification," *Security Technol. Design*, vol. 19, 1997.

[81] N. Evans, S. Marcel, A. Ross and A. B. J. Teoh, "Biometrics security and privacy protection [from the guest editors]," *IEEE Signal Processing Magazine*, vol. 32, no. 5, pp. 17–18, 2015.

[82] J. D. Woodward, "Biometrics: Privacy's foe or privacy's friend?" *Proceedings of the IEEE*, vol. 85, no. 9, pp. 1480–1492, 1997.