SAPIENZA
UNIVERSITÀ DI ROMA

Safe and controllable information consumption for data market applications: A solution based on Trusted Execution Environments and the Ethereum blockchain

Faculty of Information Engineering, Computer Science and Statistics
Master's Degree in Computer Science

Valerio Goretti
ID number 1811110

Advisor
Prof. Claudio Di Ciccio

Co-Advisor
Prof. Sabrina Kirrane

Academic Year 2021/2022

Safe and controllable information consumption for data market applications: A solution based on Trusted Execution Environments and the Ethereum blockchain

Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: valerio.goretti98@gmail.com

Contents

Chapter 1

Introduction

Data trading has emerged as one of the most significant markets in recent years. Decisions and the economy in our world are driven by data. Data owners are exposed to a variety of privacy risks in this situation [14]. Due to this, it was necessary that decentralised projects such as Solid [3] and Digi.me [5] seek to increase data owners' control over their information, while at the same time giving smaller companies and individuals access to information that is typically controlled by centralised platform providers. Two separate solutions are being developed by Solid and digi.me. The Solid community is working to develop the tools necessary for the creation and integration of decentralized applications based on linked data concepts. Digi.me is developing the tools and technology that enable people to download their data from centralized systems and store it in an encrypted personal data store. Special tools created by Digi.me will utilise this encrypted data to operate straight from the user's personal encrypted data store. The use of these platforms enhances and expands the data market but also provides more security for the data owners. These solutions guarantee access to data by monitoring conditions imposed by users in order to preserve their data. These conditions are monitored continuously to ensure high reliability.

The following lines will give examples of studies conducted utilizing blockchain technology to provide users more control over their personal data online [41]. In their paper, Ayoade et al. [7] propose a framework for blockchain applications to access evidence collected in a trusted execution environment (TEE). This enables them to control and track data access. A secure usage control system for digital rights management with the internet of things was introduced by Zhaofeng et al. [43]. The system is implemented using a blockchain for data and IoT management. DistU, a distributed usage control paradigm for industrial blockchain frameworks, is conceptualized by Khan et al. [18]. It includes processes for continuous monitoring that enable it to control a resource as it is being used and modify attributes

accordingly, performing various actions like denying or revoking permissions. In a data market scenario, Xiao et al. [40] introduce PrivacyGuard, a system that gives data owners control over who can access and use their personal data.

In comparison with the state of the art, we can consider blockchain a suitable data structure for a data control context it allows to control access to data and all the changes inherent in data management. Thanks to these features, in many works where it interacts with Solid, blockchain serves as the central character. Ramachandran et al. [32] and Cai et al. [12] propose security mechanisms from the perspectives of authentication and signatures using blockchain technology in a Solid environment. Becker et al. [11] show how a blockchain-based payment system can be used to trade the data stored in Solid pods as a marketplace. In contrast to the solution proposed in this thesis, in the study just shown, access to the data through usage control is not considered, therefore, data owner is unable to choose how its data is to be used.

The studies just presented by Xiao et al. and Becker et al. propose a data market idea that makes use of a TEE. The TEE provides a vulnerability-free environment with full control over the hardware used. This environment facilitates the storage of data and utilisation of resources in a policy-compliant manner. Leveraging this technology is essential in order to control the devices on which sensitive data will be managed and so ensure that all data access criteria are respected at all times. Data owners are able to maintain control over their data even after data consumers have gotten copies of their data thanks to the implementation of usage control through blockchain applications and TEE, which further supports the Solid concept of data ownership.

1.1 Contribution

My thesis work focuses on trusted applications running in a TEE for resource utilization in a blockchain-based data marketplace. The goal of the thesis is to employ the security features of TEEs to establish a safe and fair resource utilization standard for information retrieved from the market. The TEE layer manages access to the market while guaranteeing security over other users' data. In the data market, users can either access the data of other users, assuming the role of consumer user, or upload their own data, assuming the role of owner user. The TEE layer is related to the data consumer side. In the data market, for each resource initialized, the owner users can define rules regarding the resource's access and use. These rules must be fulfilled by consumer users who want to use the information resource. To this end, the TEE is utilized.

1.2 Structure of the thesis

The thesis is divided into 7 chapters that explain the research and development efforts made to create a framework for decentralized data market administration through the use of a TEE. The Chapter 1 gives a general introduction to the thesis. In the Chapter 2, a general overview of the necessary topics for system design was given. The Chapter 3 presents the use case of the data market in detail. Chapter 4 presents the system's architecture and design. Implementation information is provided in Chapter 5. Chapter 6 includes an explanation of the evaluations performed and the results. The conclusions reached as a result of the evaluation and the possibilities for expansion of the project are described in Chapter 7.

Chapter 2

Background

2.1 Distributed Ledger Technology - Blockchain

The term DLT stands for **Distributed Ledger Technologies**. The DLT is a network composed of known and verified nodes and serves as a shared database between them. Many times the term DLT and blockchain are interchanged, but this is not entirely accurate. **Blockchain** is a data structure that is used for the permanent storage of transactions, while DLT indicates a data structure that resides on multiple devices and is therefore geographically distributed. Furthermore, it does not determine the existence of a cryptocurrency or require mining (the cryptocurrency mining process used in blockchains) and this is another distinguishing feature [10]. Thus, we can say that blockchain is a subset of DLT [17] and the first to study them was Nakamoto [26]. Blockchain is a distributed, decentralized, and immutable ledger used to maintain transaction history. All network participants can access the data, which is regularly appended [6]. A blockchain, at its heart, is a chain of linked blocks that are linked together by hash codes and each block contains a reference to the one before it. Due to the link between the blocks, altering the content of one block would cause the invalidation of the following ones. This makes it difficult for anybody to alter the content [15]. The fundamental parts of blockchain are, transactions, blocks, and consensus mechanisms. Transactions are the information stored within the network nodes. Blocks are collections of validated transactions. Consensus mechanisms are algorithms used for the verification of transactions and their insertion into the blockchain [19]. There are two categories of blockchains: **permissioned** and **permissionless**. In permissionless blockchains, the user is able to read and write on the network without the need for authorisation. In blockchain permissioned, an entity assigns network users the roles and operations they can perform. In particular, it decides whether a user is enabled to write or read on the blockchain [39].

Blockchains provide a set of **features** that make them useful in many contexts, including immutability, transparency, traceability and automation. Once the data is placed on the blockchain, it is **immutable**. This is made possible by the cryptographic method that binds the blocks within the blockchain. Considering public, or permissionless, blockchains, users can read all transactions within the network. This provides a great **transparency** feature. The data within the blockchain, in addition to being immutable, are forever available for viewing once written within a block [42]. In order to restrict access to all resources while maintaining the transparency of the blockchain, Marangone et al. [24] propose a solution according to encrypting data on the basis of user permissions. In this way, only those who have access to the data are actually able to decrypt it. Blockchains also have a high degree of automation. This automation is given by the possibility of executing pieces of code within the blockchain. These pieces of code are called "smart contracts" and are defined for the first time by Szabo [37]. Smart contracts are executed when certain conditions are met. Smart contracts are executed to automate certain processes where both participants in the contract already agree what the result of the processing is [44]. Given the immutability characteristic of the blockchain, it guarantees the **traceability** of stored data. In recent years, blockchain has been widely used and studied in the context of supply chains, and studies show that its use in this area improves performance thanks to the traceability of the manufactured product [13, 8, 16, 21, 22]. Tracking can be essential to monitor certain parameters detected during the creation process. Being a transparent data structure, data acquired through the use of sensors are exposed to be monitored and keep track of parameters and production progress. The use of sensors in a blockchain context enables the automation of the acquisition and storage of information ensuring continuity.

2.2 Trusted Execution Environment

The **Trusted Execution Environment** (TEE) is a tamper-proof processing environment that runs on a separation kernel. The TEE is a combination of both software and hardware features that isolates the execution of code from the operation environment [25]. The separation kernel is the fundamental element to ensure a separate execution between two environments. It was first introduced in [33] and allows multiple systems requiring different levels of security to coexist on one platform. Thanks to this kernel, the system is divided into several partitions, guaranteeing strong isolation [34]. The TEE guarantees the **authenticity** of the code it executes, the **integrity** of the runtime states and the **confidentiality** of the code, data and runtime states stored on the persistent memory. The content generated by the TEE

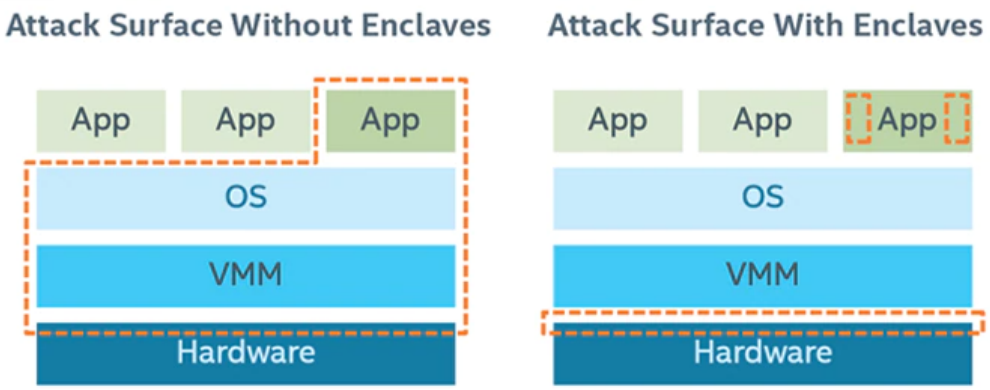

Figure 2.1. Attack-surface areas with and without Intel Software Guard Extensions enclaves

Source: https://www.intel.com/content/www/us/en/developer/articles/training/intel-software-guard-extensions-tutorial-part-1-foundation.html

is not static. The data are updated in a secure manner. Thanks to this feature, the TEE resists both software and hardware attacks, making even backdoor security flaws impossible to exploit [34]. There are many providers of TEE that differ in the system on which they are executed. In this work, the TEE provided by Intel was used, this technology is called **Intel Software Guard Extensions** [1](Intel SGX). In Intel SGX terminology, private information (e.g. passwords, encryption keys, etc.) destined for access by only one person are called application secrets. Intel SGX is a set of CPU-level instructions that allow applications to create enclaves. An enclave is a protected area of the application that guarantees the confidentiality and integrity of the data and code within it. These guarantees are also effective against malware with administrative privileges. The code is built as a Windows Dynamic Link Library (DLL) file. The use of one or more enclaves within an application makes it possible to reduce the potential attack surfaces of an application. The benefits that an enclave brings to an application can be seen in Fig. 2.1.

Going into more detail, an enclave cannot be read or written to from outside the enclave, only the enclave itself can change its secrets, independent of the CPU (Central Processing Unit) privileges used. Indeed, it is not possible to access the enclave by manipulating registers or the stack. Every call made to the enclave needs a new instruction which performs checks aimed at protecting the data that are only accessible through the enclave code. The data within the enclave, in addition to being difficult to access, is encrypted. An access to the DRAM (Dynamic Random Access Memory) modules would only obtain the encrypted data. The

cryptographic key changes randomly each time the system is rebooted following a shutdown or hibernation. One problem with protected memory is its size. Typically, protected memory ranges between 64MB and 128MB. When the data within the protected memory is heavier, the memory space is moved to the **untrusted** area of the application putting data to risks. The unsafe zone is one of the two main components of an SGX-based application. An application using Intel SGX consists of a trusted and an untrusted component. We have seen that the trusted component is the enclaves. The untrusted component is the remaining part of the application. The trusted part of the application has no possibility of interacting with any other external component except the untrusted part. Therefore, enclaves cannot communicate directly with third-party components. Nevertheless, the fewer interactions between the trusted and untrusted part, greater will be the security guaranteed by the application. When more than one enclave is used in an application, or more than one application makes use of enclaves, it is necessary to authenticate the enclaves before proceeding with data exchange. In the Intel SGX architecture, **attestations** serve this purpose. There are two types of attestations: **local attestation** and **remote attestation**. Local attestation is useful when an application uses more than one enclave which must cooperate to perform a task, or two enclaves from two different applications must exchange data. Each must confirm to the other that it is trusted in order to create a secure session for exchanging data. To ensure its security, a session key is exchanged that is used to encrypt the data within the session. Obviously, since it is not possible for one enclave to access the memory area of the other, the data must be deferenced outside the enclave and marshalled in the other enclave. Remote attestation involves the generation of a value that is sent to the two parties to establish trust. This value is generated by a combination of hardware and Intel SGX software. More specifically, the value is generated through the combination of information from the Intel SGX-enabled CPU and from some software provided by Intel SGX (the enclave to be authenticated, Quoting Enclave, QE, and the Provisioning Enclave, PvE)). This data is sent through an authenticated channel to a remote server, which determines whether the enclave in question was created by an authentic Intel SGX and thus whether it is trusted. Once both parties are authenticated, a secure channel is created through which information can be exchanged. As mentioned above, the data within the enclave are encrypted. The encryption of this data is done through the process of **data sealing**. This process allows data to be written to untrusted memories without exposing their contents. Data sealing can be done in 2 ways, using the **enclave identity** and using the **seal identity**. Sealing to the Enclave Identity allows the creation of a unique key for the enclave. Once changes are made to the enclave signature, the key is changed, making

the data inaccessible to the new version of the enclave. When the second type of sealing is used, sealing to sealing identity, multiple enclaves can seal and unseal data. This allows enclaves to unseal data sealed by earlier versions of the same enclave as well as share the seal between multiple applications. With some care, it is possible to use this method of sealing data by sharing it with other applications but at the same time preventing unsealing of data from earlier versions of the enclave. The addition of an enclave version number to the enclave's signature would make it impossible to unseal the data from future versions of the enclave.

2.3 Usage Control

Access control is concerned with preventing illegal access to computational resources and digital information. The goal of access control is to maintain control over computational resources and digital information in order to avoid unauthorised disclosure (confidentiality) and inappropriate malicious modifications (integrity), while allowing authorised entities access (availability) [20]. Usage control performs controls during data access management compared to the other data access models. Usage control was introduced in the early 2000s and was considered an evolution of access control [27]. The idea behind usage control is to monitor how data is used. A data provider's owner must establish a usage control policy that specifies the conditions that a data consumer who receives a copy of the provider's data must meet [31]. The most established usage control model was developed by Park and is called UCON [28]. This section will continue the discussion based on the UCON model in order to best illustrate the characteristics of usage control. UCON model is a generalisation of access control that includes obligations, conditions, mutability and continuity controls. UCON encompasses the integration of access control, trust management, and digital rights management. In comparison to typical access control policies and models, UCON offers finer-grained control over the use of digital objects by combining these three domains. UCON model consists of three main components and three additional components. The core components are subjects, objects and rights. The additional components are authorization rules, conditions, and obligations. The authentication phase in UCON must include the authentication rules but may include all additional components. **Subjects** are entities that claim ownership of an object. These entities also have attributes that are required during authentication. Subjects are models of users in the system. A subject can be a group or an individual and has the right to objects. When we speak of a group, we mean a collection of users who have the same rights (if they have equal roles) to the same objects. In UCON, subjects can take on the roles of

consumer subjects, provider subjects and identifiee subjects. Consumer subjects are those who receive access to a resource through rights. Subject providers are those who provide the rights to access their own data. Identified subjects are subjects that are identified in digital objects and include all sensitive information. **Objects** are the entities managed and maintained by subjects. Objects have attributes that can be used during authentication and can be categorised into classes so that access can be granted to an entire class and not to all individual files. Objects may be privacy sensitive or privacy non-sensitive. A privacy-sensitive object includes information relating to the identification of a subject and therefore potentially dangerous to expose the subject's privacy. In UCON there are also derived objects, which are copies of objects once the rights over them have been obtained. A subject may have privileges over objects. These privileges are called **rights**, which allow subjects access to objects. Rights are also divided into consumer rights, provider rights and identifiee rights. When we talk about rights, in UCON there are access rights and delegation rights. Rights are divided into many categories according to what they enable the receiving user to do (view, modification etc.). **Authorisation Rules** are a set of requirements to be fulfilled in order to make it possible for a subject to access an object. There are two types of authorisation rules, rights-related authorisation rules and obligation-related authorisation rules. The first type is used to see whether a subject can exercise rights over an object. The second type is used to check whether a person has consented to an obligation that must be fulfilled after obtaining or using rights to a digital object. **Conditions** are a set of decision criteria that the system must check together with the authorisation rules during the authorisation process. There are two types of conditions: dynamic conditions and static conditions. Dynamic conditions include information that must be verified each time it is used due to updates, e.g. a condition on the number of accesses. Static conditions include information that does not have to be checked each time it is used because it is not frequently updated, e.g. a condition on the location or time of access. Dynamic conditions are stateful and static conditions are stateless. Rules of authorisation and conditions are different. Authorisation rules are a set of decision criteria used to verify whether a subject can use the rights to an object, while a condition is used to verify whether restrictions on access to and use of an object are met. **Obligations** are mandatory requirements that a party must perform after obtaining or exercising rights over an object. Before receiving rights, a user accepts the obligations he or she will have to fulfil. Obligations will be controlled by authorisation rules [29]. Park e Sandhu once laid the basis of usage control. They improved the model first by defining a family of ABC models as a core model for usage control in paper [36] and then further refining it by creating the UCON_{ABC} model [30].

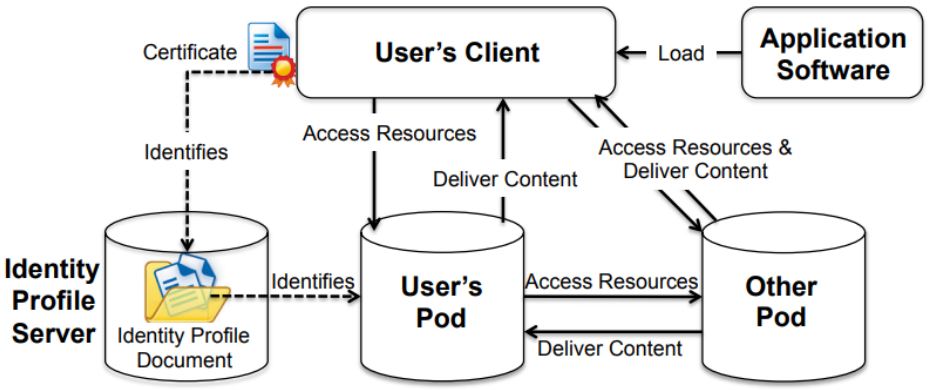

Figure 2.2. Solid Architecture

Source: [35]

2.4 Solid Web

Solid is a decentralized platform for social Web applications [23]. Solid project [3] facilitating the decentralisation of web data. Tim Berners-Lee, the creator of the World Wide Web (WWW), is leading the project proposing to change the WWW as we know it to a decentralised version in order to increase control over the data exchanged over the web. In the Solid Web, people can store their data securely online through the use of Pods (Personal Online Datastore).

Figure 2.2 represents the architecture of Solid from this, we can see that each user can have one or more pods, and Solid-based applications can access these pods through well-defined mechanisms, such as decentralised authentication and access control mechanisms that guarantee user privacy. Within the Solid platform, users check their identity using an RDF profile document. This verification will allow the user to interact with their pods to access the data. The user will need to obtain a Solid application from an application provider in order to utilize it, and it will communicate with them in order to access the pods. The data in Solid are managed by a `REST` architecture via `APIs`. When a user uploads data, it is either stored through the use of an `HTTP POST` call within a container or the URL is stored through an `HTTP PUT` call. Furthermore, a deletion is handled through an `HTTP DELETE` call while an update is performed through an `HTTP PUT` or `HTTP PATCH`. Performing an `HTTP GET` on a container returns an enumeration of the elements present in the container. Data in Solid is identified through a Uniform Resource Identifier (URI). In addition, Solid distinguishes data into two types: structured data (represented with Resource Description Framework, RDF) and unstructured data (for example video, images etc.). Solid applications employ `RESTful HTTP` operations to read and write

data stored in users' pods. Solid servers might provide optional SPARQL support. Applications can define complicated data retrieval procedures, including those that call for server-to-server communication via link-following SPARQL, on servers that support it. This facilitates the creation of Solid applications by allowing developers to delegate complex, multi-pod data retrieval tasks to the server. Since the Solid pod servers are application-agnostic, new applications can be created without modifying the servers [35].

Chapter 3

Motivating Use Case Scenario

The use case of the study is a decentralized data market that aims to facilitate access to data. This marketplace involves the subscription of users who want to use it. Once subscribed, users will be able to upload their data and choose how this data is to be used by consumer users and what rules they have to comply with in order to use it. Subscribed users can also request the data of another user. When requesting a data item in the marketplace, the system checks whether the user who made the request fulfills the requirements set by the data owner in order to allow in order to allow access and use of the resource. The following lines describe the use case example depicted in Fig. 3.1. Technical details will be explained in Chapter 4.

The main actors of the example are Alice and Bob, who do not know each other. Alice is a research biologist in the area of zoology, she is currently conducting a study on animal taxonomy. Bob is a photographer with a passion for animals. Bob decides to collect some of his shots in an album and publish it so that other people can also enjoy the photos. Bob chooses to subscribe to the data marketplace and pays the registration fee. From this moment, Bob can start uploading his data into the marketplace (point 1 in the figure). Once the data has been uploaded, Bob can choose to set rules for the use of his data by setting: purpose of data use, expiration date, maximum number of possible accesses and geographical location where the data is available. Bob decides that he only wants his images to be used for the purpose of scientific research, setting a maximum number of 100 accesses and an expiry date of 20 days after the retrieval of the data. Furthermore, Bob decides to make the data available anywhere in the world by not setting geographical limitations. Alice needs to find material to continue her study and decides to subscribe to the data market in order to look for some useful material. Once the subscription has been paid, Alice starts searching for material in the data market and finds Bob's photo album. Alice decides to request a copy from the market (point 2 in the figure). The data market will retrieve the actual position of the data within Bob's memory (point

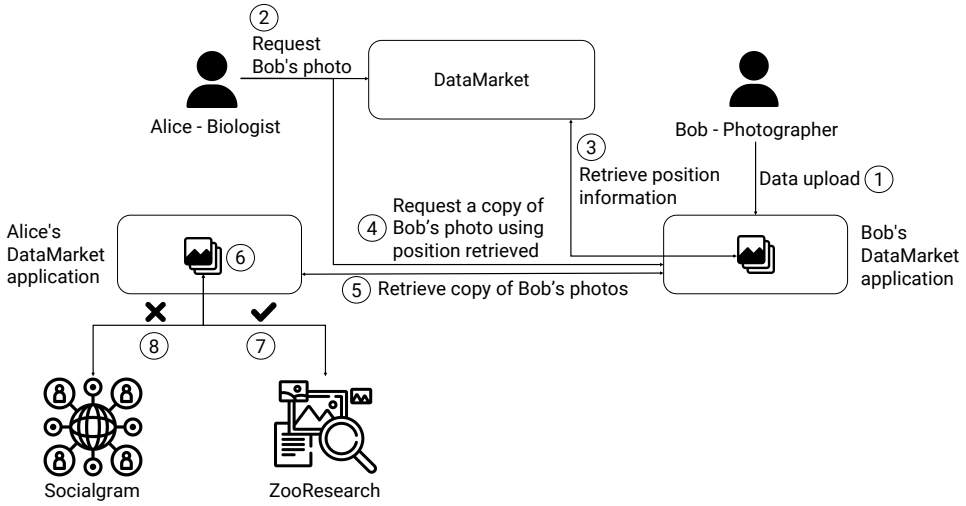

Figure 3.1. Use case example flow

3 in the figure) and enable Alice to make the request. Once she has requested the data to Bob (point 4 in the figure), a copy of the photos is sent to a trusted part of the application in Alice's device (point 5 in the figure). This trusted part will monitor the use of the retrieved data according to the rules that Bob has set. Once retrieved, Alice chooses to open these photographs through a famous application in the research area used for analysing zoological images called "ZooResearch". Since the purpose of this application (research) corresponds with the data usage rule set by Bob, and no geographical limitations have been chosen, Alice is able to use the photos (point 7 in the figure). At the end of the working day, Alice is particularly surprised by a picture uploaded by Bob and decides to share it on her social channels with her friends. To do this, she decides to use the "Socialgram" application, a popular social network. Alice opens Socialgram and tries to access the copy of the data she has in its memory but without success (point 8 in the figure). Indeed, the purpose of the Socialgram application (social), is not among the data expected purposes and thus she will not be able to publish the photo. The purpose of the trusted part in Alice's device is to prevent undesired use by the owners of the data retrieved from the market. In addition, this part ensures to data owners like Bob that once the data expires, in this case after 20 days, or if the number of permitted accesses is exceeded, in this case 100 accesses, the data is completely removed from the consumers' device.

Chapter 4

Design

In order to address the needs described in the motivating use case scenario, this section shows the architecture of the data market. The architecture is divided into 3 main parts: the blockchain, the pod and the trusted application and it is shown in Fig. 4.1. Components will be described in the following sections. Since the study aims to manage decentralised market information in a controlled and safe manner, after setting the scene and presenting the bigger picture, this chapter will focus on the architecture of the TEE.

4.1 Pod

Pods are archives of personal data used by users. In the architecture shown, they use the pod to load data into the marketplace and choose which obligation rules must be observed in order to access it. In short, the pod is a kind of file system dedicated to the use of usage control. A user can choose to restrict consumer use of their uploaded resource by setting a maximum number of accesses and a validity period for the data. This last condition is valid from the moment the consumer requests and receives the information. The data owner can choose a maximum duration in number of days, which is converted into an expiration date at the time of the consumer's request. In addition, the owner of the data can choose the geographical location and usage domain in which to make it available. Once the data has been received, the consumer users will store it within their trusted application (which will be discussed in more detail in Section 4.3). The user will then be able to use this data through applications that will request it. To be able to use the data, the application requesting access must meet the conditions of geographical location and domain. For a better understanding of the concept of domain of use, please refer to the example in Chapter 3.

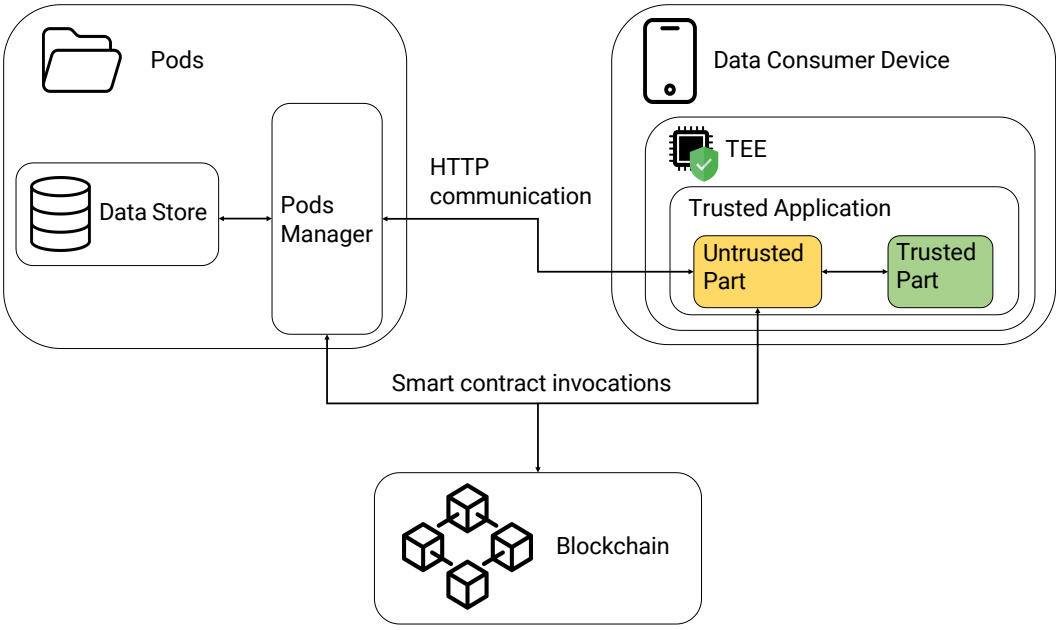

Figure 4.1. Data Market Architecture

4.2 Blockchain

In the data market, blockchain is used to index and track all uploaded resources and some metadata related to data. Blockchain is used to implement data market logic. When a user uploads data through their pod, the data will be stored in the pod but it will be identified and indexed in the blockchain, storing the location of the data within the pod, access rules and additional metadata useful for accessing to the data. In addition to resource data, subscription data is also stored on the blockchain. When users subscribe to the platform, they are allowed to offer their own data for sale by storing it on the pod and to request the data of other people through the trusted application. Thus, the blockchain is used by both the pod and the trusted application to retrieve information about resources in the market and how to manage them. In addiction it is used to complete and authorise users to perform requests between the trusted application and pods. When a user wants to retrieve data from another user before requesting it from the pod, they need to know exactly where it is located. The trusted application will be used to retrieve the exact location of the resource within the pod by performing a request to the blockchain, which will check the subscription of the consumer user and if successful will provide the information needed to retrieve the data. In addition, pods can request information from the blockchain to manage their data. Pods and trusted application communicate with the blockchain and vice versa via blockchain oracles for invoking smart contracts.

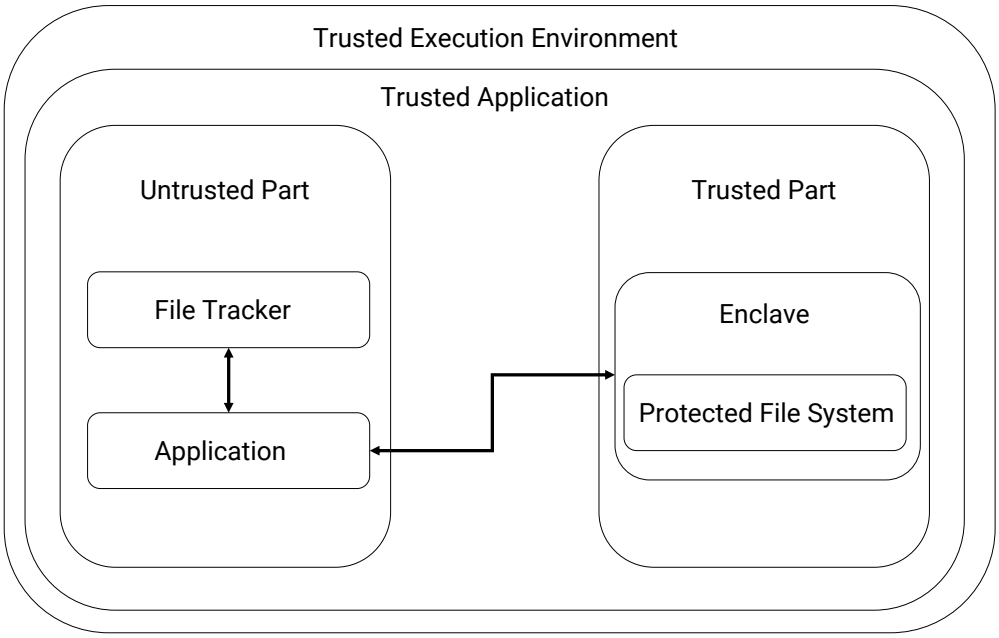

Figure 4.2. Trusted application architecture

4.3 Trusted Application

The trusted application is at the core of the thesis work. A trusted application is an application that is executed on a TEE. The TEE consists of hardware and software and is used to guarantee protection of the data stored within it in order to provide an isolated execution of the trusted application and consequently provide high data confidentiality. In the context of the data market, trusted application is used by data consumers to make requests for resources in which they are interested. Through the trusted application, marketplace users can retrieve data from other users' pods in a controlled manner. The entire retrieval process is controlled by the blockchain that provides and controls the data exchange. The proposed market infrastructure assumes that a copy of the data requested by other users is stored within the trusted application in order to monitor its use and check that the rules set by the data owner are always fulfilled.

4.3.1 Structure of the Trusted Application

Once the purpose of the TEE and the trusted application has been clarified, we move on to analyse the design of the trusted application. Figure 4.2 focuses on the architecture of the TEE, going into more detail about its specifications.

The trusted application consists of two fundamental parts: the trusted part, also called the enclave, and the untrusted part. The trusted part cannot communicate

directly with the outside world. Every information that enters or leaves the trusted part passes through the untrusted part. Since the trusted application is used to retrieve and share data, we need the untrusted part of the application to make the trusted part interact with the outside world. When data needs to be retrieved from another user's pod, the trusted application makes a call to the blockchain to retrieve the exact location of the data in which the user is interested. The blockchain verifies that the requesting user has paid for the marketplace membership and then, if everything is in order, forwards the location of the resource to the untrusted part. Once the trusted application has received the location of the data, it will make a request to the received location (the exact position of the pod and the resource within it). The pod will accept the request and forward the data to the untrusted part of the trusted application, which will store the usage rules chosen by the data owner (more details in Section 4.3.3) and forward the resource to the trusted part to create a copy for use.

4.3.2 Encryption in the Trusted Application

Inside the trusted part, the data is encrypted according to processor-generated keys. In the data market context, in this part the data retrieved by the consumer will be stored. The trusted part's data will be encrypted and only accessible with the use of the enclave. Additionally, only Intel-created libraries or standard libraries that Intel has wrapped in the trusted part may be used; external libraries are not permitted. Otherwise, in the untrusted part, third-party libraries can be used and the data is not encrypted. Data may be at risk since it must pass through the untrusted part. In order to prevent data from being left in the untrusted part of the application without encryption, the architecture includes an encryption scheme described in Fig. 4.3 and Fig. 4.4. When the trusted application is started, the untrusted part through dedicated Intel SGX functions starts the enclave, which is deleted each time the application is closed and recreated later. During the creation of the application's trusted part, the public and private keys of the application are generated in the trusted part. The private key of the application is only stored in the trusted part, while the public key is stored in both parts of the application. Having the public key stored in the untrusted part allows the trusted application to make requests for new market resources without making requests each time to the trusted part. The flow of requests for new data in the market is depicted by Fig. 4.3. When users decide to request a resource on the market, in addition to the request, they also send the application's public key with which the data will be encrypted. In this way, the data arrives encrypted with the application's public key to the trusted part, which with the private key will be able to decrypt the data and store

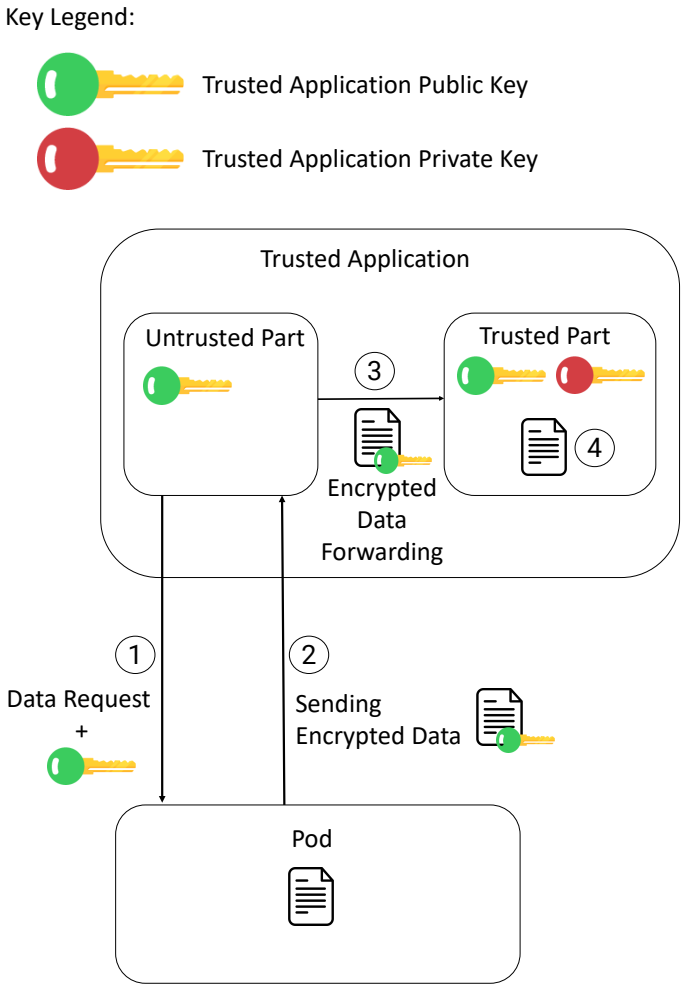

Figure 4.3. Encryption flow when the trusted application retrieves data from a pod

it. An application may then need to access data in the trusted part. This situation is depicted in Fig. 4.4. In order to preserve the data from the trusted part towards the application, the application also sends its public key within the request. The untrusted part of the application checks that the application can actually request the resource (the checks made by the untrusted part will be explained in Section 4.3.3), and forwards the request to the trusted part, which uses the external application's public key to encrypt the requested data and forward it back to the untrusted part, which will send it to the application. The application will then be able to decrypt the data using its private key.

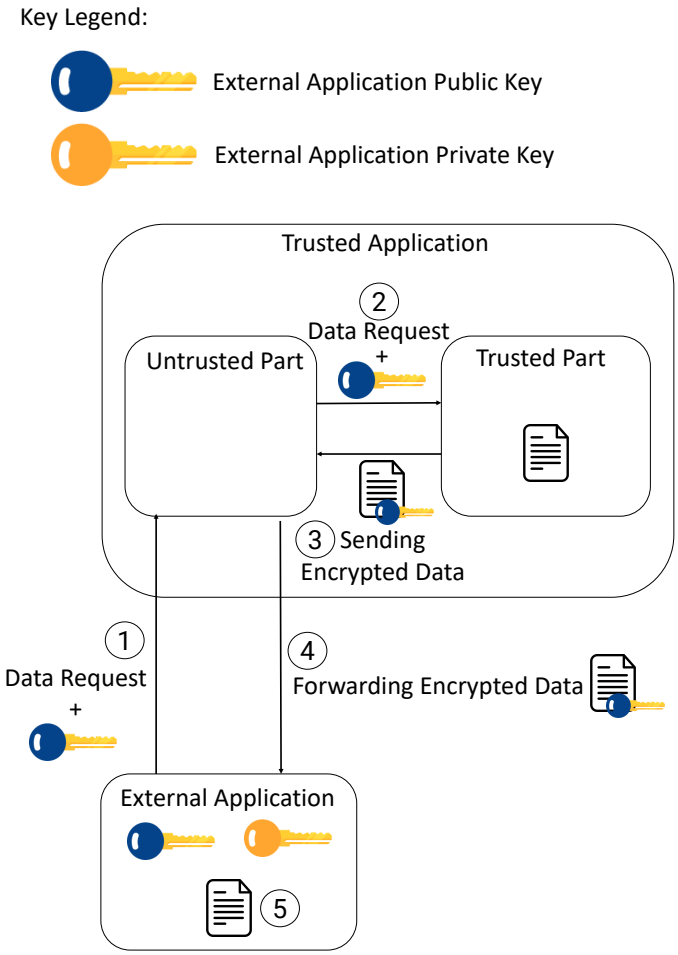

Figure 4.4. Encryption flow when an external application requests data from the trusted application

4.3.3 Checks performed by the untrusted part

The untrusted part of the application serves both to interact with the outside world and thus interface the trusted part with applications, pods and the blockchain, as well as to handle incoming requests. As can be seen in Fig. 4.2, there is a module called FileTracker in the untrusted part. This module is used as a kind of log for managing data access. In the FileTracker, the applications that have consent to make requests to the data in the enclave and all related information are stored. Also stored in the FileTracker is the list of data in the trusted part and the relevant conditions to be fulfilled in order to access the data. In this way, we are able to reduce the interactions between the trusted part and the untrusted part. Guaranteeing non-sensitive data immediately ready to be accessed. Applications submit in the

data request both their public key with which the data will be encrypted (as seen in Section 4.3.2) and their current location. In this way, the untrusted part will start by checking whether the application has received authorisation to access resources, and thus whether it is on the list. Next, if it has authorisation, the purpose of the application is checked. Finally, the current geographical location is checked and compared with the location limitation provided by the data owner. Should one of these checks fail, access to the resource is not permitted. Once it has checked that the application complies with all the criteria set by the owner of the data, the untrusted part goes on to check that the maximum number of accesses has not yet been exceeded and also checks that the data has not expired. If the resource has not yet reached the maximum number of accesses and has not expired, it is sent to the application.

Chapter 5

Implementation

For the implementation, I resort to **Intel SGX** [1] and **Web3.js** [4]. Intel SGX is used to develop the TEE. Intel offers this software to create trusted applications. Intel allows the enclave to be programmed in both native C and C++, while the interaction between the enclave and the untrusted part must be in C. As Fig. 5.1 shows, there are two possibilities for developing an application on a TEE. These solutions differ in the choice of framework for the user interface. Using a C#-based framework, since it is only possible to interact with the enclave via the C programming language, involves the use of two additional layers in order to build bridge functions to utilise the enclave. Instead, choosing a native C or C++ application allows the application to be programmed close to the enclave. For the work performed in this thesis, I have chosen to develop the trusted application by developing it entirely in native C. As already mentioned in Section 4.3.2, Intel SGX does not allow the use of classic C libraries in the trusted part, but wrappers created by Intel are used for security reasons. In this way, the system does not rely on third-party libraries, which could be dangerous for such a system. Communication between the untrusted part of the application and the blockchain is allowed through the use of Web3.js. which is a collection of libraries allowing interaction with Ethereum via HTTP, IPC or WebSocket. The blockchain used during the thesis work is provided by Ganache and is an Ethereum blokchain. Web3.js therefore allows communication between the untrusted part of the application and the Ethereum blockchain. This section will focus on the implementation of the trusted application as the thesis work is focused on this part of the market. The next subsections will explain the basic steps of the code. For more information, the code can be found on the Github repository at the link https://github.com/ValerioGoretti/TEE.

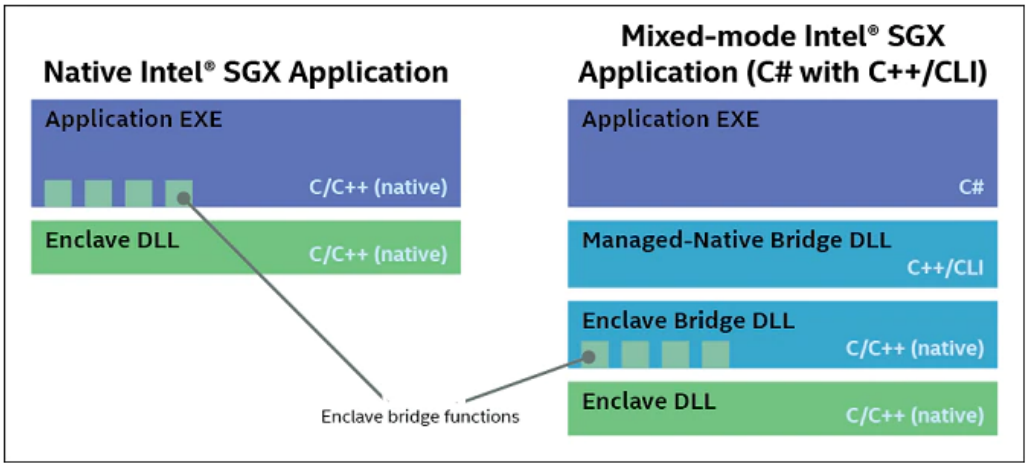

Figure 5.1. Minimum component structures for native and C# Intel SGX applications.

Source: https://www.intel.com/content/www/us/en/developer/articles/training/intel-software-guard-extensions-tutorial-part-2-app-design.html

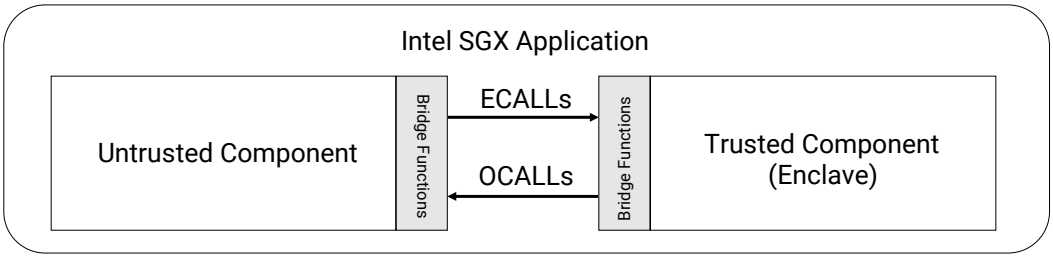

Figure 5.2. Interaction structure between the trusted and untrusted component of an Intel SGX Application

5.1 Trusted Part

5.1.1 Enclave.edl file

Enclave development begins with the creation of the Enclave.edl file where edl stands for 'Enclave Definition Language'. The edl files define the bridging functions between the trusted part and the untrusted part. The file is divided into three parts. In the first part, other edl files relating to the Intel SGX libraries that will be used are imported. In the case of our application, the package for the Protected File System Library [2] was imported on line 3 of Listing 5.1. The second part is used to define the ECALLs, the functions contained within the enclave, in prototype form. In the third part we define the prototypes of the OCALLs functions, these are the functions within the untrusted part that are called during the execution of some function in the trusted part. It is easy to see that our enclave has no OCALLs and the ECALLs

Direction	ECALL	OCALL
in	The buffer is copied from the application into the enclave. Changes will only affect the buffer inside the enclave.	The buffer is copied from the enclave to the application. Changes will only affect the buffer outside the enclave.
out	A buffer will be allocated inside the enclave and initialized with zeros. It will be copied to the original buffer when the ECALL exits.	A buffer will be allocated outside the e nclave and initialized with zeros. This untrusted buffer will be copied to the original buffer in the enclave when the OCALL exits.
in, out	Data is copied back and forth.	Same as ECALLs.
user_check	With user_check attribute the raw pointer address will be passed and the programmer should do the bounds checking on the address if needed.	

Table 5.1. Pointer direction parameters and their meanings in ECALL and OCALL

are defined from line 5 to line 16 of Listing 5.1. The term “Ecall” stands for Enclave Call and means a call made by a function to the enclave. The term “OCALL” (Out Call) refers to a call from inside the enclave to the untrusted component. As shown in Fig. 5.2, Ecall and Ocall are used for the interaction between the trusted and untrusted part of the application. In edl, parameters are handled differently. If it is a variable, no specification is necessary as it is copied into the memory stack of the enclave. In the case of pointers, on the other hand, since one is working on a memory cell, it is necessary to indicate the size of the buffer and in which direction the data is to be saved. The possible choices regarding directions are: ’in’, ’out’, ’in, out’ and ’user _check’. Descriptions of these attributes can be found in Table 5.1.

```
enclave {
    from "sgx_tstdc.edl" import *;
    from "sgx_tprotected_fs.edl" import *;

    trusted {
        /* define ECALLs here. */
      public void getSecret([out,size=len] char* buf,size_t len);
        public void setSecret([in, string] char* setString);

        /* function of IPFS - Intel Protected File System*/
        public void ecall_enclaveString([out, size=len] char *s, size_t
            len);
      public SGX_FILE* ecall_file_open([in,string]const char*
          filename,[in,string]const char* mode);
      public size_t ecall_file_write([user_check]SGX_FILE* fp, [in] char
          data[100]);
      public size_t ecall_file_read([user_check]SGX_FILE*
          fp,[in,out,string]char* readData);
      public int32_t ecall_file_close([user_check]SGX_FILE* fp);
      public int32_t ecall_file_delete([in, string]char* filename);
    };

    untrusted {
        /* define OCALLs here. */

    };
};
```

Listing 5.1. Contents of the Enclave Definition Language file

5.1.2 Enclave.cpp file

Enclave.cpp file concerns the programming of the enclave and thus the trusted part of the application. The file contains the functions declared in the prototypes of the Enclave.edl file. In the following lines, the main functions within the enclave.cpp file will be described. In the context of the data market, the main use of the enclave is to manage data from the market. In fact, we see that most of the functions in the enclave are used for file management. In Listing 5.2, the code for the functions **ecall_file_open** and **ecall_file_write** is shown. The first function is used to open files in the enclave. In fact, it takes as parameters the name of the stored file and the mode in which it is to be opened, and returns an element of type SGX_FILE, which is a data type defined within the Protected File System Library.

The ecall_file_write function is used to create a copy of the file from outside the trusted application. It takes an object of type SGX_FILE and the data to be written within the file and proceeds with the writing using functions from the Protected File System Library such as sgx_fwrite. The function returns the number of elements written within the file, which is useful as the char array pointed to by the 'data' pointer may have a very large size.

```
1  SGX_FILE* ecall_file_open(const char* filename, const char* mode)
2  {
3     SGX_FILE* a;
4     a = sgx_fopen_auto_key(filename, mode);
5     return a;
6  }
7
8  size_t ecall_file_write(SGX_FILE* fp, char *data)
9  {
10    size_t sizeofWrite;
11    size_t len = strlen(data);
12    sizeofWrite = sgx_fwrite(data, sizeof(char), len, fp);
13
14    for (int i = 0; i < 5; i++)
15    {
16       char buffer[] = { 'x' , 'c' };
17       sizeofWrite += sgx_fwrite(buffer, sizeof(char), sizeof(buffer), fp);
18    }
19
20    return sizeofWrite;
21 }
```

Listing 5.2. ECALL functions for opening and writing a file in the enclave

In Listing 5.3, we can see the last functions of the enclave needed to manage the protected file system. Among these functions, we have: **ecall_file_read**, **ecall_file_close** and **ecall_file_delete**. The first function is used to read the contents of a file within the enclave. The content read remains inside the enclave, returning the size of the file read. The second function is used to close open files. Function ecall_file_delete is used to delete files stored within the enclave. This function in the context of the data market is essential to remove files that have expired or have the maximum number of uses.

```
size_t ecall_file_read(SGX_FILE* fp, char* readData)
{
   char* data;
   uint64_t startN = 1;
   sgx_fseek(fp, 0, SEEK_END);
   uint64_t finalN = sgx_ftell(fp);
   sgx_fseek(fp, 0, SEEK_SET);

   data = (char*)malloc(sizeof(char) * finalN);
   memset(data, 0, sizeof(char) * finalN);

   size_t sizeofRead = sgx_fread(data, startN, finalN, fp);
   int len = strlen(data);
   memcpy(readData, data, sizeofRead);
   return sizeofRead;
}

int32_t ecall_file_close(SGX_FILE* fp)
{
   int32_t a;
   a = sgx_fclose(fp);
   return a;
}

int32_t ecall_file_delete(char* filename)
{
   int32_t a;
   a = sgx_remove(filename);
   return a;
}
```

Listing 5.3. ECALL functions for reading, closing and deleting files in the enclave

5.2 Untrusted Part

The untrusted part is based on a Windows Console Application through which the user can interact via a menu and choose the operations to be performed. In Fig. 5.3 we see the application's menu, and by typing in one of the numbers indicated in the menu, the application will perform the request. Then, by pressing button 1, the user can see the list of files stored within the trusted part. By pressing key 2, he can request a data item and this will be stored in the trusted part. By pressing key

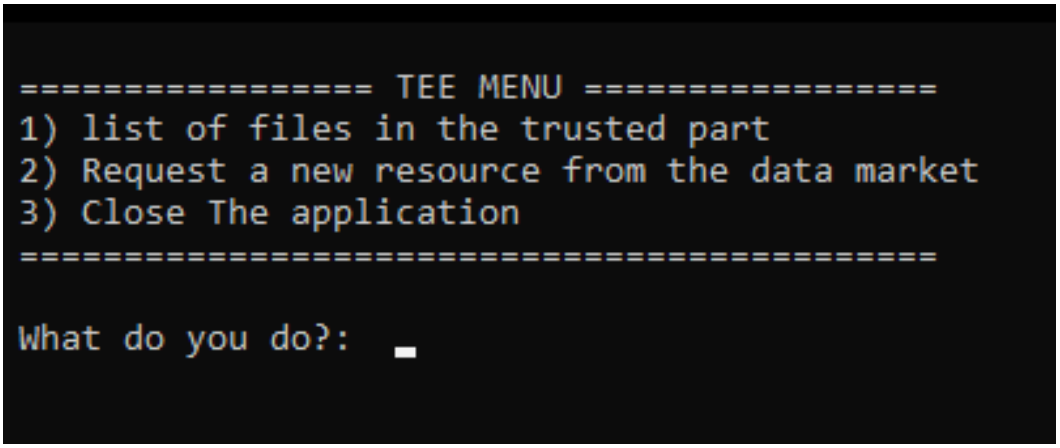

Figure 5.3. Console application execution

3, the user will exit the application by destroying the enclave for security reasons, which will be recreated when the application is launched again.

Having seen the user-side functioning of the trusted application, the next few lines will describe some of the functions in the untrusted part. The functions that will be described are used for interaction with the outside world (blockchain and pod) and the management of the trusted application's data. The first functionality of the untrusted part to be illustrated is that of resource access. The function is called resourceAccess and is visible in Listing 5.4. In the context of the data market, this function is called by applications that want to access a resource contained in the trusted part. The function takes as input the identifier of the resource the application wishes to access, the application's identifier and its current location. Once this information has been defined, the function checks whether the requested resource is present within the trusted memory. If it is not present, the request will produce no file and return an error. Otherwise, the function will proceed to determine whether the application respects the usage criteria such as the domain and the location from which it is requesting access. Only if the application meets both of these criteria the function will go on to check the other rules set by the data owner. Among the criteria strictly related to the file, the function first proceeds to check whether the file has expired or not. If the file has not expired, the function proceeds to check whether the maximum number of accesses has been reached or can be accessed. If all checks are successful, the maximum access counter of the file is decreased and the data is returned to the application that requested it. Otherwise, an error is generated.

```
1  FILE resourceAccess(sgx_enclave_id_t eid, char* id, char* idApp, char*
       location) {
2     FILE* file, file2;
3     char err[256];
4     errno_t e = fopen_s(&file, "FILEPATH1", "r");
5     errno_t e2 = fopen_s(&file2, "FILEPATH2", "w+");
6     if (file == NULL || file2 == NULL) {
7        strerror_s(err, 100, e);
8        printf("Unable to open file, the error is: %s", err);
9     }
10    else {
11       char buf[1000];
12       while (fgets(buf, 1000, file) != NULL) {
13          char* p = split(buf, 0);
14          if (strcmp(id, p) == 0) {
15             char* fname = split(buf, 1);
16             char domain[MAX_BUF_LEN];
17             strcpy_s(domain, MAX_BUF_LEN, split(buf, 2));
18             if (checkApplication(domain, location, idApp)) {
19                int day = atoi(split(buf, 5)), month = atoi(split(buf, 6)),
                      year = atoi(split(buf, 7));
20                if (checkDate(day, month, year) == 0) {
21                   printf("Expired File");
22                   removeFile(eid, fname);
23                }
24                else {
25                   int newA = atoi(split(buf, 3));
26                   if (newA <= 0) {
27                      printf("Maximum number of accesses reached");
28                      removeFile(eid, fname);
29                   }
30                   else {
31                      newA--;
32                      char mA[100], d[5], m[5], y[10];
33                      sprintf_s(mA, 100, "%d", newA);
34                      sprintf_s(d, 5, "%d", day);
35                      sprintf_s(m, 5, "%d", month);
36                      sprintf_s(y, 10, "%d", year);
37                      putsOnFile(file2,id,split(buf,1),
38                      split(buf, 2),mA,split(buf, 4),d,m,y);
39                   }
40                }
41             } else {
```

```
42              fputs(buf, file2);
43            }
44         } else {
45            fputs(buf, file2);
46         }
47      }
48      fclose(file);
49      fclose(file2);
50      if (remove("FILEPATH") == 0)
51         printf("\nFile removed\n");
52      else
53         printf("Unable to delete the file");
54      rename("FILEPATH2", "FILEPATH1");
55   }
56   return retriveFile(id);
57 }
```

Listing 5.4. Code of the resourceAccess function, which allows access to a resource in the enclave

The second functionality illustrated in this subsection concerning the untrusted part of the application is the one called following a request by the consumer user for a file on the market. The functions used for this task are those illustrated in Listing 5.5, addFile function and the createNewFileInEnclave function. Starting with the addFile function, which after a request for a file from the marketplace, the pod forwards the contents of the file and all the rules to be respected from the outside world. The function stores these rules set by the owner in the untrusted part. More specifically, an id is stored for the file, the name of the file, the usage domain to be respected, the location where the resource is available and the expiry date of the file. The addFile function prepares to include this newly received file in the list of files contained within the trusted part. To store the contents of the file within the enclave, the function createNewFileInEnclave is called, which takes as input the contents of the file (encrypted as described in the design phase, see Section 4.3.2) and the name of the file within which it will be stored, and interacts with the enclave to proceed with the creation of the file. This function involves 3 calls to the enclave with the aim of writing a new file to it. The function creates a pointer to an SGX_FILE and instantiates the mode in which to open this file, in write mode and in the case where it is not present, we create it. This data is passed to the ECALL ecall_file_open. We then pass to the ECALL ecall_file_write the content to be placed inside the file and other information needed to write the file we have just opened. Once the execution of this ECALL is complete, the file is created and

stored within the enclave. The contents of the file are passed unencrypted and stored in the enclave since, after creation, the file is automatically re-encoded by another Intel SGX automatic encoding procedure by which only the enclave can access that resource. Finally, the file is closed via the ECALL ecall_file_close.

```
1  void createNewFileInEnclave(sgx_enclave_id_t eid, char* buffer, char*
       filename) {
2     SGX_FILE* fp;
3     sgx_status_t ret = SGX_SUCCESS;
4     const char* mode = "w+";
5     //File opening
6     ret = ecall_file_open(eid, &fp, filename, mode);
7     //Write buffer value into the file
8     size_t sizeOfWrite = 0;
9     ret = ecall_file_write(eid, &sizeOfWrite, fp, buffer);
10    //file Closure
11    int32_t fileHandle;
12    ret = ecall_file_close(eid, &fileHandle, fp);
13 }
14
15 void addFile(sgx_enclave_id_t eid, char* buffer, char* id, char* filename,
       char* purpose, int maxAccess, char* location, int day, int month, int
       year) {
16    FILE* file;
17    char err[256];
18    errno_t e = fopen_s(&file, "FILEPATH", "a");
19
20    if (file == NULL) {
21       strerror_s(err, 100, e);
22       printf("Unable to open file, the error is: %s", err);
23    }
24    else {
25       char mA[100], y[10], d[5], m[5];
26       sprintf_s(mA, 100, "%d", maxAccess);
27       sprintf_s(y, 10, "%d", year);
28       sprintf_s(m, 5, "%d", month);
29       sprintf_s(d, 5, "%d", day);
30       putsOnFile(file,id,filename,purpose, mA, location,d,m,y);
31       fclose(file);
32    }
33    createNewFileInEnclave(eid, buffer, filename);
34 }
```

Listing 5.5. Code for the functions addFile and createNewFileInEnclave, which are used to store the copy of the file from the market in the enclave

In conclusion, code that enables an application to request permission from a trusted application is shown. The code is presented by Listing 5.6 and is called by an

application the first time it makes a call to the trusted application. The application inserts the necessary data into the addApplication function within the call. The function takes as input the name and domain of the application and stores them in the untrusted part. In this way, following a second call, the trusted application can check whether the application requesting the data has been authorised and in which domain it is working. The location of the application is required when requesting access to the file (required in the resourceAccess function, see Listing 5.4) as it may change over time.

```
1  void addApplication(char* name, char* domain) {
2     FILE* file;
3     char err[256];
4     errno_t e = fopen_s(&file, "FILEPATH");
5
6     if (file == NULL) {
7        strerror_s(err, 100, e);
8        printf("Unable to open file, the error is: %s", err);
9     }
10    else {
11       char newline[3] = "\n";
12         AllowApplication(file,name,domain);
13       fclose(file);
14    }
15 }
```

Listing 5.6. Code of the addApplication function which authorises the request of resources to applications

Chapter 6

Evaluation

In order to evaluate the solution produced, I chose to perform a verification by design. This evaluation is approached by proof the strength through objective evidence that specified requirements have been fulfilled. The requirements chosen to be proven are essential for the realisation of a good trusted application. The requirements chosen are: only the enclave can decrypt the data (R1), nobody except the enclave can access the data (R2) and at the time of its expiration, the data no longer exists for anyone (R3).

R1 was chosen because it is essential that the data encoded within the TEE be readable only by the enclave in order to guarantee the confidentiality of the information and to allow use in only controlled scenarios. Requirement R2 is a nuance of Requirement R1 but it is necessary to distinguish them. In fact, R2 requires that the system allow access to the data, both encrypted and unencrypted, only to the enclave, therefore, that external components cannot access it. Requirement 3 is the one most related to the use case and requires that after an expiration or exhaustion of possible accesses, the data is no longer present in the trusted application.

The R1 requirement is demonstrable since within the trusted application, only the enclave stores the private key to decrypt the data. When the enclave is created, the untrusted part of the application only know and store the public key, unlike the enclave will store both. Only the untrusted part of the application can communicate with the trusted part. No other application, even if it has full privileges, can do this (even another enclave in the same trusted application) and if within the untrusted part the data is encrypted with the application's public key and the private key is only stored within the enclave, only the enclave can decrypt the data. In addition, the private key, like all data in the enclave, is encrypted with Intel SGX's automatic encryption. Therefore, the enclave memory is encrypted using industry-standard encryption algorithms with replay protection. The key with which data is encrypted by Intel SGX is stored inside the CPU and is unreadable. This key is randomly

generated each time the enclave is recreated. This additional encryption also makes a direct attack on the memory or DRAM modules impossible since the attacker will access encrypted data.

In order to prove the R2 requirement, we must remember the features explained in the R1 demonstration regarding encryption performed by Intel SGX. In fact, to make its data available, since only it can decrypt it, the enclave provides APIs to call its functionality (ECALLs). Thanks to ECALLs, the programmer of the enclave can trace its boundaries; without them, the enclave and the data it contains could not be accessed. So this shows us that untrusted elements cannot access the data stored in the trusted part except through the enclave. So only the enclave can access its own data.

Finally, to prove requirement R3, as already explained in the design and implementation chapters. When one of the data invalidating conditions occurs, which are the maximum number of accesses reached or the date of the data has passed, the data is automatically removed from the enclave. This means that both the enclave and the untrusted part cannot access or retrieve the data in any way. The memory used for storing that data is freed.

Chapter 7

Conclusion and Future Work

The work carried out for this thesis allowed us to study and enter the world of TEEs, evaluating their potential and essentiality in certain types of contexts, such as the data market. Although an initial version of the TEE was developed, with this one we had control of the entire application, ensuring correct use of the data while maintaining the grantie given by the usage control. Thus, a data owner user knows for sure that once the rules of use have been set, those rules will be observed thanks to a trusted application that continuously monitors them. In addition to the monitoring work provided by the trusted application, the use of blockchain provides the system with decentralisation and traceability of data and requests within the market. The project is open to future developments that will improve some of the limitations present in the thesis work.

One of the most important future implementations for the project is the addition of the enclave attestation, more precisely the remote attestation provided by Intel. Attestation is the process of proving that an enclave has been established on a trusted platform so that it is recognisable and trusted during interactions. It is possible to establish local attestation to identify two enclaves on the same platform but we are interested in remote attestation that uses a remote provider to attest trust. Since the trusted application has to communicate with many external parties, we can effectively attest that the parties are trusted by establishing a session and, during the establishment, both parties attest and then a secure data exchange can begin.

Then the approach to the development of the trusted application could be changed by incorporating a mixed approach using C#, which would allow us to implement a more user-friendly GUI, which would guarantee a better interaction with the consumer user who wants to request data in the market. Changing the approach increases the distance between the application code and the native C enclave code. The current enclave code can be maintained, but we would need to

create the bridge layers in order to develop the C# application.

Finally, I plan to expand the evaluation of the system by adding a quantitative evaluation to the qualitative evaluation of the system.

In order to do this, I intend to evaluate the system using the measurement tools studied by Bailleu et al. in their work [9]. These researchers have developed a model called TEE-Perf which is an application and platform independent performance measurement tool for TEE. The tool consists of four parts: compiler, recorder, analyser and visualiser. The compiler injects the profiling code into the application. The recorder is used to set components to track the execution of the code. Through the data tracked by the recorder, the tool supports the measurement of "Call Stack", "Queries" and "Selective code profiling". The "Call Stack" measurement is used for call reconstruction, thanks to which we can check the call time of even the most complex queries. "Queries" allows us to search for contention in the code or call dependencies that lead to high overhead. "Selective code profiling" allows us to analyse the code only in certain parts and not in its entirety.

A second evaluation will be carried out following the work performed by Weichbrodt et al. [38]. From this work, Sgx-perf was designed, a set of tools for analysing TEE performance and behaviour, indeed, SGX-Perf is composed of several tools working together. The tools are: an event logger, a working set estimator and an analyser. These tools make it possible to monitor the behaviour of the TEE via logs without modifying the sgx SDK code. Using sgx-Perf tools, we can track incoming calls to the enclave and outgoing calls through an incoming and an outgoing wrapper code that monitor the movements of function calls. Thanks to these wrappers, it is also possible to evaluate call execution times. It is also possible to trace the dependencies between threads within the enclave. Another tool will allow us to calculate the enclave's performance by considering Asynchronous Enclave Exits (AEX). This aims to calculate the actual execution time of the enclave, leaving out what happens outside. If there is a problem outside, a metric that classically calculates performance is to the detriment of the enclave, whereas this one manages to separate the two and give enclave-related judgement. Through a final tool, it is also possible to check the amount of memory used. Enclaves use a memory space called Enclave page cache (EPC). EPCs are limited between 64MB and 128MB of memory. When the memory space exceeds this, the EPC is moved to the unsafe area of the application. This tool then monitors the movement between the safe and unsafe zones of the pages. Being able to determine which parts of the enclave are less used. Enclaves should be designed in such a way that pages are not moved to the non-secure zone. In this regard, it is possible through one of the Sgx-Perf tools to monitor the performance of pages in the TEE. Once these behaviours have

been monitored, Sgx-Perf provides an analysis of the data obtained and gives the developer advice on how to improve the application's performance.

In conclusion, in a world like ours that relies on data and structures it in every possible way, the use of a trusted execution environment coupled with a blockchain can curb data abuse. The case study of the data market is a clear example of how these problems, through the correct use of technology, can be managed.

Bibliography

[1] Intel® Software Guard Extensions. https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/software-guard-extensions.html, accessed: November 18, 2022

[2] Overview of Intel Protected File System Library Using Software Guard Extensions. https://www.intel.com/content/www/us/en/developer/articles/technical/overview-of-intel-protected-file-system-library-using-software-guard-extensions.html#:~:text=IntelÂő%20Protected%20File%20System%20Library%20provides%20protected%20files%20API,API%27s%20provided%20by%20Intel%20SGX., accessed: November 18, 2022

[3] Solid. https://solidproject.org/about, accessed: November 18, 2022

[4] web3.js - Ethereum JavaScript API. https://web3js.readthedocs.io/en/v1.7.5/, accessed: November 18, 2022

[5] What is digi.me? https://digi.me/what-is-digime/, accessed: November 18, 2022

[6] Antonopoulos, A.M.: Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc." (2014)

[7] Ayoade, G., Karande, V., Khan, L., Hamlen, K.: Decentralized iot data management using blockchain and trusted execution environment. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI). pp. 15–22 (2018)

[8] Azzi, R., Chamoun, R.K., Sokhn, M.: The power of a blockchain-based supply chain. Computers & industrial engineering **135**, 582–592 (2019)

[9] Bailleu, M., Dragoti, D., Bhatotia, P., Fetzer, C.: Tee-perf: A profiler for trusted execution environments. In: 2019 49th Annual IEEE/IFIP International

Conference on Dependable Systems and Networks (DSN). pp. 414–421 (2019). https://doi.org/10.1109/DSN.2019.00050

[10] Bashir, I.: Mastering blockchain. Packt Publishing Ltd (2017)

[11] Becker, H., Vu, H., Katzenbach, A., Braun, C.H., Käfer, T.: Monetising resources on a solid pod using blockchain transactions. In: The Semantic Web: ESWC 2021 Satellite Events. pp. 49–53 (2021)

[12] Cai, T., Yang, Z., Chen, W., Zheng, Z., Yu, Y.: A blockchain-assisted trust access authentication system for solid. IEEE Access (2020)

[13] Chang, S.E., Chen, Y.: When blockchain meets supply chain: A systematic literature review on current development and potential applications. IEEE Access **8**, 62478–62494 (2020)

[14] Dai, W., Dai, C., Choo, K.K.R., Cui, C., Zou, D., Jin, H.: Sdte: A secure blockchain-based data trading ecosystem. IEEE Transactions on Information Forensics and Security **15**, 725–737 (2019)

[15] El Ioini, N., Pahl, C.: A review of distributed ledger technologies. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 277–288. Springer (2018)

[16] Francisco, K., Swanson, D.: The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency. Logistics **2**(1) (2018). https://doi.org/10.3390/logistics2010002, https://www.mdpi.com/2305-6290/2/1/2

[17] Hellwig, D., Karlic, G., Huchzermeier, A., et al.: Build your own blockchain. Springer (2020)

[18] Khan, M.Y., Zuhairi, M.F., Syed, T.A., Alghamdi, T.G., Marmolejo-Saucedo, J.A.: An extended access control model for permissioned blockchain frameworks. Wirel. Networks **26**(7), 4943–4954 (2020)

[19] Lashkari, B., Musilek, P.: A comprehensive review of blockchain consensus mechanisms. IEEE Access **9**, 43620–43652 (2021)

[20] Lazouski, A., Martinelli, F., Mori, P.: Usage control in computer security: A survey. Computer Science Review **4**(2), 81–99 (2010)

[21] Li, Z., Wu, H., King, B., Miled, Z.B., Wassick, J., Tazelaar, J.: A hybrid blockchain ledger for supply chain visibility. In: 2018 17th International Symposium on Parallel and Distributed Computing (ISPDC). pp. 118–125. IEEE (2018)

[22] Longo, F., Nicoletti, L., Padovano, A., d'Atri, G., Forte, M.: Blockchain-enabled supply chain: An experimental study. Computers & Industrial Engineering **136**, 57–69 (2019)

[23] Mansour, E., Sambra, A.V., Hawke, S., Zereba, M., Capadisli, S., Ghanem, A., Aboulnaga, A., Berners-Lee, T.: A demonstration of the solid platform for social web applications. In: Proceedings of the 25th international conference companion on world wide web. pp. 223–226 (2016)

[24] Marangone, E., Di Ciccio, C., Weber, I.: Fine-grained data access control for collaborative process execution on blockchain. arXiv preprint arXiv:2207.08484 (2022)

[25] McGillion, B., Dettenborn, T., Nyman, T., Asokan, N.: Open-tee–an open virtual trusted execution environment. In: 2015 IEEE Trustcom/BigDataSE/ISPA. vol. 1, pp. 400–407. IEEE (2015)

[26] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)

[27] Park, J., Sandhu, R.: A unified framework for next generation access control. George Mason University **100** (2006)

[28] Park, J., Sandhu, R.: A position paper: a usage control (ucon) model for social networks privacy (2000)

[29] Park, J., Sandhu, R.: Towards usage control models: beyond traditional access control. In: Proceedings of the seventh ACM symposium on Access control models and technologies. pp. 57–64 (2002)

[30] Park, J., Sandhu, R.: The uconabc usage control model. ACM transactions on information and system security (TISSEC) **7**(1), 128–174 (2004)

[31] Pretschner, A., Hilty, M., Basin, D.: Distributed usage control. Communications of the ACM **49**(9), 39–44 (2006)

[32] Ramachandran, M., Chowdhury, N., Third, A., Domingue, J., Quick, K., Bachler, M.: Towards complete decentralised verification of data with confidentiality: Different ways to connect solid pods and blockchain. In: Companion Proceedings of the Web Conference 2020. p. 645–649 (2020)

[33] Rushby, J.M.: Design and verification of secure systems. ACM SIGOPS Operating Systems Review **15**(5), 12–21 (1981)

[34] Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: what it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/ISPA. vol. 1, pp. 57–64. IEEE (2015)

[35] Sambra, A.V., Mansour, E., Hawke, S., Zereba, M., Greco, N., Ghanem, A., Zagidulin, D., Aboulnaga, A., Berners-Lee, T.: Solid : A platform for decentralized social applications based on linked data (2016)

[36] Sandhu, R., Park, J.: Usage control: A vision for next generation access control. In: International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security. pp. 17–31. Springer (2003)

[37] Szabo, N.: Formalizing and securing relationships on public networks. First monday (1997)

[38] Weichbrodt, N., Aublin, P.L., Kapitza, R.: sgx-perf: A performance analysis tool for intel sgx enclaves. In: Proceedings of the 19th International Middleware Conference. pp. 201–213 (2018)

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

[40] Xiao, Y., Zhang, N., Li, J., Lou, W., Hou, Y.T.: Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) Computer Security – ESORICS 2020. pp. 610–629 (2020)

[41] Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications (2019)

[42] Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., Rimba, P.: A taxonomy of blockchain-based systems for architecture design. In: 2017 IEEE international conference on software architecture (ICSA). pp. 243–252. IEEE (2017)

[43] Zhaofeng, M., Lingyun, W., Xiaochang, W., Zhen, W., Weizhe, Z.: Blockchain-enabled decentralized trust management and secure usage control of iot big data. IEEE Internet of Things Journal **7**(5), 4000–4015 (2020)

[44] Zou, W., Lo, D., Kochhar, P.S., Le, X.B.D., Xia, X., Feng, Y., Chen, Z., Xu, B.: Smart contract development: Challenges and opportunities. IEEE Transactions on Software Engineering **47**(10), 2084–2106 (2019)