A FULLY DECENTRALIZED FRAMEWORK FOR SECURELY SHARING DIGITAL CONTENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMET SERHAT DEMİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF CYBER SECURITY

SEPTEMBER 2019

Approval of the thesis:

A FULLY DECENTRALIZED FRAMEWORK FOR SECURELY SHARING DIGITAL CONTENT

Submitted by AHMET SERHAT DEMİR in partial fulfillment of the requirements for the degree of **Doctor of Philosophy/Master of Science in Cyber Security Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics** _____

Assoc. Prof. Dr. Aysu Betin Can
Head of Department, **Cyber Security** _____

Asst. Prof. Dr. Aybar Can Acar
Supervisor, **Health Informatics Dept., METU** _____

Dr. Ali Arifoğlu
Co-Supervisor, **Information Systems Dept., METU** _____

Examining Committee Members:

Prof. Dr. Ali Aydın Selçuk
Computer Engineering Dept., TOBB ETÜ _____

Asst. Prof. Dr. Aybar Can Acar
Health Informatics Dept., METU _____

Assoc. Prof. Dr. Banu Günel Kılıç
Information Systems Dept., METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Ahmet Serhat Demir

Signature : _____

ABSTRACT

A FULLY DECENTRALIZED FRAMEWORK FOR SECURELY SHARING DIGITAL CONTENT

Demir, Ahmet Serhat

MSc., Department of Cyber Security

Supervisor: Asst. Prof. Dr. Aybar Can Acar

Co-Supervisor: Dr. Ali Arifoğlu

September 2019, 97 pages

Blockchain is a secure, immutable, and distributed public ledger that stores transactional data. It enables information transfer without the need for a trusted third party via its decentralized consensus mechanism. Besides finance, blockchain technology has the potential to change several industries, through smart contracts and decentralized applications. This thesis explores using blockchain technology, smart contracts, and Ethereum Web 3.0 stack for secure information and file sharing on a fully decentralized architecture. We aim to remove the need for a central authority in all layers of the application, and thus provide an alternative to the drawbacks of centralized content exchange platforms. Accordingly, a proof-of-concept decentralized application is designed. This design is implemented in the Ethereum ecosystem using blockchain for the immutable distributed ledger, Ether for cash transfers, and smart contracts for application logic. Since data storage in blockchain is expensive, Swarm is used as a decentralized reliable content storage system. Nevertheless, permissionless systems in the Ethereum ecosystem lack necessary data privacy, which causes risk for secure information exchange. In order to provide access control for sensitive content delivery without the need for a pre-shared secret, public key encryption is used. Also, to enable fair exchange between separate untrusted rational parties, a double escrow functionality is implemented as a model of fairness via incentive with deposit penalties. According to the validation and evaluation of our proof-of-concept, we show that Ethereum Web 3.0 stack is applicable to securely sharing and exchanging digital content without relying on a trusted third party.

Keywords: blockchain, smart contract, decentralized application, digital content sharing, fair exchange

ÖZ

TAMAMEN MERKEZİ OLMAYAN GÜVENLİ DİJİTAL İÇERİK PAYLAŞIM SİSTEMİ

Demir, Ahmet Serhat

Yüksek Lisans, Siber Güvenlik Bölümü

Tez Yöneticisi: Dr. Öğr. Üyesi Aybar Can Acar

Ortak Tez Yöneticisi: Öğr. Görevlisi Ali Arifoğlu

Eylül 2019, 97 sayfa

Blokzincir, merkezi olmayan konsensüs özelliği sayesinde güvenilir bir üçünü tarafa ihtiyaç duymadan bilgi aktarımı sağlayan, işlem verilerinin kaydedildiği, güvenli, değişmez, dağıtık bir genel defterdir. Akıllı sözleşmeler ve merkezi olmayan uygulamalar ile blokzincir teknolojisi finans sektörünün yanı sıra, diğer birçok sektörü de derinden etkileme potansiyeline sahiptir. Bu tez, blokzincir teknolojisi, akıllı sözleşme ve Ethereum Web 3.0 yığınının tamamen merkezi olmayan bir mimaride dosya ve bilgi paylaşımı için kullanılabilirliğini incelemektedir. Sürecin tüm katmanlarında merkezi otorite ihtiyacını ve bu sayede merkezi içerik paylaşım platformlarının sakıncalarını ortadan kaldırmak amaçlanmıştır. Bu doğrultuda, kavram kanıtlama amaçlı bir merkezi olmayan uygulama tasarlanmıştır. Bu tasarım Ethereum ekosistemi içinde, değişmez dağıtık defter olarak blokzincir, para transferi için Ether, uygulama mantığı için de akıllı sözleşmeler kullanılarak geliştirilmiştir. Blokzincirde veri kaydetmek pahalı bir işlem olduğundan, merkezi olmayan güvenilir depolama sistemi Swarm kullanılmıştır. Bununla beraber, izin gerektirmeyen Ethereum ortamı gerekli veri mahremiyetini sağlamadığından, güvenli bilgi alışverişi için risk oluşturmaktadır. Güvenli bir içerik paylaşımı için erişim denetimi sağlamak amacıyla, genel anahtar şifreleme sistemlerinden yararlanılmıştır. Ayrıca, birbirlerine güven duymayan akılcı taraflar arasında adil bir alışveriş sağlamak amacıyla iki taraflı emanet özelliği eklenmiştir. Geliştirilen kavram kanıtlama uygulaması için gerçekleştirdiğimiz testlere ve yapılan değerlendirmelere göre, Ethereum Web 3.0 yığınının güvenilir üçüncü bir kişiye ihtiyaç duymadan güvenli bir şekilde dijital içerik paylaşımı ve alışverişini mümkün kıldığı gösterilmiştir.

Anahtar Sözcükler: blokzincir, akıllı sözleşme, merkezi olmayan uygulama, dijital içerik paylaşımı, adil alışveriş

To My Family

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my supervisor Asst. Prof. Dr. Aybar Can Acar for his guidance and advices throughout each phase of the thesis. Dr. Acar has been a positive and supportive mentor during my study.

I would like to acknowledge my co-supervisor Dr. Ali Arifoğlu for his support and inspiration in development of innovative technologies.

I would also like to thank my girlfriend and all my friends for being supportive to me.

Finally, I would like to thank to my family for always being there for me. I am most grateful to them. Without them, this work would not have been possible. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABI	Application Binary Interface
DApp	Decentralized Application
DASP	Decentralized Application Security Project
DNS	Domain Name Service
DoS	Denial of Service
ECC	Elliptic-curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ENS	Ethereum Name Service
EOA	Externally Owned Accounts
EVM	Ethereum Virtual Machine
FR	Functional Requirement
GDPR	General Data Protection Regulation
ICO	Initial Coin Offering
IoT	Internet of Things
IPFS	Interplanetary File System
NFR	Non-functional Requirement
OWASP	Open Web Application Security Project
PKC	Public Key Cryptography
PoC	Proof of Concept
PoS	Proof-of-Stake
PoW	Proof-of-Work
TTP	Trusted third party

CHAPTER 1

INTRODUCTION

Blockchain technology has become one of the most popular and significant technologies recently, along with Artificial Intelligence (AI) and Internet of Things (IoT). Blockchain is basically an immutable, decentralized, and cryptographically secure distributed database (or "ledger"). When the Bitcoin [1] blockchain was first introduced, cryptocurrencies were seen as the only innovation blockchain technology would bring, and consequently would challenge only the current financial system. However, with the introduction of Ethereum [2] as a smart contract and decentralized application platform, the true potential of blockchain technology was realized. Accordingly, blockchain systems can function as a way to transfer any "value" across the internet [3]. Using smart contracts, which are self-executing computer programs that runs on the blockchain network, blockchain technology has made it possible to enable business transformations in several domains. The main innovation that blockchain technology offers is eliminating the necessity of trusted third parties (TTPs) and centralized authorities that exist in the conventional business models. This is made possible with decentralized consensus mechanisms.

Blockchain provides a new type of internet infrastructure based on decentralized applications that operate on a distributed network, and maintains a source of information sharing [4]. Also, using blockchain (permissionless or permissioned) is suitable when it is desired that numerous mutually mistrusting parties be able to interact and change the state of the system, without the need for a TTP [5]. This is achieved because instead of a central authority, blockchain relies on network consensus for decision making and value exchange. It follows that, blockchain technology can be used to enable separate parties share and exchange information without relying on a TTP, in a decentralized way.

1.1. Problem Statement

The privacy of the blockchain data is not addressed in permissionless blockchains, such as Bitcoin and Ethereum. As anybody can join the network, and the data is available to any peer of the network. This yields a problem when users want to share or exchange sensitive information or digital content. As well as sharing digital content, since these blockchains also include built-in cryptocurrencies, it is logical to use these currencies to sell digital content too, like an e-commerce system. Naturally, the

aforementioned privacy issue becomes a problem to be solved in order ensure that only the buyers can access the content that is for sale. We used public key cryptography (PKC) as an access control mechanism and solve the lack of privacy of the sensitive information in the blockchain.

Publishing digital content is often achieved via simple mechanisms. However, exchanging value in the internet, especially in the field of e-commerce where users exchange goods for money, requires more complex protocols. Such protocols should satisfy *goods atomicity* [6], where a seller get paid only when the buyer receives the good, and a buyer receives the good only when the seller gets paid. This refers the fairness of the exchange in e-commerce. According to Asokan [7], an exchange is fair when either both parties get the item they expect, or none of them gets any additional information at the end of the protocol. This requirement of fairness yields a problem called *fair exchange* between multi parties in distributed systems. Relying on TTPs is a common solution to this problem, and many conventional exchange and e-commerce platforms are dependent on TTPs to ensure fairness. Actually, it was shown that fair exchange protocols cannot completely assure strong fairness without TTPs [42]. Two party digital content exchange using blockchain and cryptocurrencies can be considered as a subset of fair exchange protocols in electronic commerce. Instead of a conventional TTP, we have utilized a smart contract as an escrow service acting as a trustless third party in order to provide a close approximation of fairness. This is achieved by incentivizing rational pairs to act honestly via predefined deposits, which they would lose if they act maliciously.

Storing optional information or unnecessary data in blockchain is inefficient in several ways. First, since every node in the blockchain stores blockchain data, the data would be replicated to all the nodes, which would inflate the blockchain data size. Second, due to the limitation on the block size in blockchain, files have to be split and reassembled off-chain [8]. Third, since read/write operations in blockchain cost Ether, i.e., actual money, it becomes very expensive to store large data in the blockchain. As a result of these, an off-chain storage system is suitable while dealing with digital content in the blockchain. In order to maintain network consensus and decentralization, it makes sense to use immutable decentralized storage systems such as IPFS, and Swarm, instead of a local or cloud storage.

By taking these issues in to consideration, this study explores the usability of blockchain technology and smart contracts for securely sharing and fairly exchanging digital content in distributed storage with a built-in payment system on a fully decentralized architecture where the parties do not need to trust each other. For this purpose, a proof of concept (PoC) was designed, implemented, tested, and evaluated.

1.2. Motivation

Starting from the 2000s, especially with the advance of social media, the internet has been considered more than just a source of information, but a space where people can contribute, publish, and share digital content using platforms such as blogs, social media, and media exchange platforms [9]. Digital content publishing and sharing over

the internet has become a giant business with millions of contributors and worth billions of dollars. This content can be any digitally stored information such as audio, image, text, software, or video files. There are numerous platforms in the internet, such as YouTube, Facebook, Shutterstock, Soundcloud, Amazon KDP, etc., where publishers share, exchange, or sell their digital content. Currently all of these platforms operate on a centralized scheme and act as, or rely on a TTP between the users to achieve fairness. On the other hand, centralization of this business model running on the internet for the purpose of sharing digital content has its own drawbacks, hence it makes sense to decentralize the digital content sharing process without relying on a TTP. In this section we address these issues:

- **Privacy issues with the centralized content sharing platforms**.

  Currently centralized platforms or marketplaces for content publishing, sharing, exchanging are controlled by a few giant companies. Enormous amount of personal data is stored in data servers of these oligopolies. As a result of very long privacy policies and user agreements, most of the time users lose control of their own data once the data is stored in the servers. Often users are required to give detailed personal information such as credit card information, address, company name etc., before they are accepted to a content publishing platform, which raises privacy issues. Although there are recently emerged laws to protect the privacy of the users, such as EU General Data Protection Regulation (GDPR), the logs of the database transactions are still managed centrally by this companies, which can raise questions about the immutability of the logs.

  Users trust their data with these platforms but sometimes their data are given to yet another party. For example Facebook shared personally identifiable information of millions of its users with Cambridge Analytica which raised many debates [10]. Besides, there is always a risk of the information on the servers being hacked, as occurred in several data breaches in the last decade [11].

- **Monetary issues with the centralized content sharing platforms**.

  Digital content exchange platforms can be free to use, or charge money for membership, or charge commission for digital content trades. When the platform is free to use, there is a renowned marketing expression: if you are not paying for the product, then you are the product. Most of the time users lose control and ownership of their content when they upload it to the internet, either receive a small amount of money or usually not paid at all, besides they need to deal with annoying advertisements. For the platforms which provide secure content trading, they get a cut out of the price from both the sellers and buyers. Because of this, sellers receive less money than they deserve, and buyers pay more money than they actually should. On top of this, content publishers have to wait a few days until they receive their money.

  In content publishing platforms where the digital asset is for sale, as an addition to the platform, financial institutions act as another TTP for the actual payment.

Current financial systems are centralized, and prone to risks of centralized architectures. They validate, safeguard, and preserve transactions between users. As a result, and to compensate for some unavoidable fraud, transaction costs are high [12]. In addition to this, international, or large money transfers take longer time. Also, since the banking system is a permissioned system, financial institutions control who can use the financial system.

- **Issues with the centralized web architecture.**

  Current web services are very much centralized and built on the client/server architecture. Clients access web services and digital content over servers such as web servers, DNS servers, cloud servers, Content Distribution Servers (CDN). However, this architecture forms a single point of failure, and is vulnerable to denial-of-service (DoS) attacks. For example, Amazon Web Services (AWS) outages occurred in 2015 and 2017, which took 5 and 4 hours respectively, made several web services inaccessible and resulted with hundreds of millions of dollars' worth of loss [13] [14]. Also in 2016, a distributed DoS (DDoS) attack on Dyn, one of the most important DNS providers, broke several DNS services and affected access to several significant web services such as Twitter, Spotify, Netflix, etc. [15].

  Considering the number of online devices and with the increase of network bandwidth, also sometimes by amplifying the packets, serious DDoS attacks of even now more than hundreds of gigabytes per second traffic rates [16] are important threats to centralized architectures. Another drawback of the centralized architecture of the web services is censorship. There are many countries censoring several web services in their countries, because of political, military, social, security, or even copyright protection reasons [17]–[19].

1.3. Purpose and Research Questions

This study aims to explore the usability of blockchain for secure content sharing. Blockchain technology and peer-to-peer networks promise solutions to inherited problems of conventional content sharing platforms due to their centralized nature, as well as the necessity of TTPs for validating transactions between users that do not necessarily know or trust each other, for both information and currency exchange. In this study we have explored the solutions for the problems stated in *Problem Statement*.

The goal of this thesis is to design, implement, and evaluate a decentralized application on the Ethereum blockchain which enables a channel for securely sharing digital content between parties that do not necessarily trust each other. The solution would remove TTPs, obtain a decentralized, immutable, secure, autonomous data storage and transaction environment with built-in currency on a permissionless distributed network underpinned by blockchain technology.

In order to reach the goal of the study, and remedy the issues with the centralized architecture that depends on TTPs, following research questions are addressed during the study:

- *Is it possible to make a secure content exchange between separate parties with payment or contract, without the need for a TTP?*

  We have investigated Ethereum blockchain technology as the underlying platform for proposed application, and smart contracts as the application logic for the secure digital content sharing decentralized application.

- *How can we combine blockchain and storage of large data in a fully decentralized way?*

  Storing data in blockchain is inefficient, so it is necessary to find a suitable way to store the large data off-chain. For this reason, we explored decentralized storage systems and how to integrate them with Ethereum blockchain and smart contracts.

- *How can we securely share content on a fully decentralized software and network architecture in a permissionless blockchain?*

  Ethereum blockchain is a permissionless and open platform, and it cannot ensure the privacy of the transactions and smart contracts. Likewise, the data in public decentralized storages are accessible by anybody. Thus, extra measures need to be taken in order to provide content confidentiality. Accordingly, we made use of PKC.

- *How can we ensure fair exchange between separated users that do not trust each other, without a conventional TTP?*

  We explored smart contracts to implement as a trustless mediator with double escrow service to achieve a reasonable approximate solution for the fair exchange problem.

The supplementary goal of this study is to address security and privacy issues of our proposed solution, as well as other non-functional requirements. In order to share content in a secure way the system should be secure, our implementation should provide confidentiality, integrity, availability of the data along with an access control mechanism. Privacy, and other aspects such as costs, scalability, usability, and performance of the proposed framework should be addressed as well. Accordingly, detailed evaluations are performed regarding these concerns, capped with discussions to reflect our point of view.

1.4. Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 covers the literature review conducted for this study. Chapter 3 outlines the necessary background information about blockchain technology and Ethereum ecosystem. Chapter 4 explores the solutions to issues stated in Section 1.1, accordingly system design and architecture of the proof of concept is sketched in this chapter. We also highlight the rationale behind the choice of technology for the design of this study. Chapter 5 deals with implementation of the designed application and its validation with a test scenario. Chapter 6 evaluates this application of the proof of concept regarding the its design requirements. We also elaborate and further survey the topic specific to each non-functional requirement. In Chapter 7, conclusion, limitations of work, and future directions are discussed.

CHAPTER 2

LITERATURE REVIEW

Blockchain has the potential to be a “disruptive” technology which makes it possible to build new business models with a built-in economic foundation. Using blockchain provides benefits such as decentralization, security, immutability, transparency. Additionally, it makes it possible to get rid of the necessity for the TTPs which act as an intermediary between users. Consequently, information and content exchange over blockchain has many benefits such as integrity, availability, and non-repudiation. There are several academic studies and industrial practices that attempt to adapt blockchain and distributed architectures for sharing digital value and content in a secure way.

In the context of this thesis, there are four technical aspects of sharing digital content. (i) Using blockchain and smart contracts to avoid TTP, (ii) utilizing decentralized storage system to avoid costly on-chain storage or centralized off-chain storage systems, (iii) enforcing access control to mitigate the lack of data privacy in Ethereum, and (iv) developing a fair exchange protocol to protect separated untrusted parties against fraud.

A permissionless blockchain, in this case Ethereum, is a transparent system. Although the transactions are cryptographically signed, blockchain data is stored in plain text and accessible to anyone due to nature of the system. This aspect of Ethereum raises privacy issues, for sensitive data in particular. In order to securely store the data in blockchain and provide privacy, extra measures need to be carried out as discussed in several works [20] [21] [22]. Cryptographically secure obfuscation [22], zero-knowledge proofs [23], using one-time accounts [23], ring signatures [24], encryption of the transactions [25], and trusted execution environments [26] are examples of approaches for this privacy issue.

2.1. Decentralized Digital Content Publishing and Sharing

There are several studies utilizing blockchains to develop a framework for sharing digital content, mostly in healthcare, in which the contents are medical records or personal health information files. However, the approaches in these studies are not as decentralized as this study aims for, and depends on some centralized authority at some point. They utilize at least one of: permissioned blockchain [27] [28] [29] [30] [31],

centralized data storage (cloud or local storage) [32] [33] [28] [29] [30], and centralized back-end logic (no smart contracts) [27] [29] [31]. The reason is that, in healthcare the data is very sensitive, there are many roles, there are various data types, and there are regulations. Hence, a high level of decentralization is harder to achieve.

The author in [34] aimed to provide a system for secure data sharing between providers and electronic health record (EHR) systems using blockchain technology. The proposed solution addresses the protection of sensitive health information in Ethereum blockchain with a containerized application that secure enables cryptographic algorithms, smart contracts and a distributed architecture for microservices. For the storage and sharing of large files, IPFS decentralized storage system was included in the implementation. The study has claimed the necessity of combining permissioned blockchain and cryptography to achieve integrity, security, and portability of user data. Accordingly, different roles have been assigned for users with different access levels in a permissioned Ethereum instance. Proposed system addresses the security and privacy issues while sharing EHR data, even so it relies on a web application layer for the management, and the design or implementation details are not discussed.

In order to have a more refined review, we further limited our interest into more decentralized study approaches for content sharing which use permissionless blockchains (i.e., Ethereum) as the distributed ledger, decentralized data storage systems (i.e., IPFS, Swarm) for content storage, and smart contracts (decentralized applications) for decentralized back-end logic.

Cui et al. [35] addressed the problems with access revocation in blockchain based file sharing when multiple users have access to the same encrypted file. Access revocation was enabled by combining blockchain with proxy re-encryption, by which a scalable key management scheme was provided. For access revocation, a re-encryption proxy re-encrypted the data while ensuring the access of the authorized users. The proposed system architecture has data owners, data users, miners with different roles, which suggests it requires a permissioned blockchain. The paper does not address the details of the implementation, for example the storage provider and back-end logic are not discussed. Nonetheless, authors addressed the access revocation issue with encrypted storage in blockchain, and promised the technical details and an implementation of a prototype as a future work, so a more decentralized approach could be assured.

The author in [36] contemplated the suitability of using blockchain technology to enable secure information sharing between multi-agency groups in crime and security domain. In order to do so, a decentralized application was developed using Ethereum blockchain, smart contracts, and a web interface, and IPFS as the off-chain storage. To share data, sender requires recipient's public address, so that she can set this account as the only authorized account in the smart contract. Sender and recipient are also required to agree on a secret key so that the confidentiality of the data is ensured with symmetric-key cryptography. The drawback of this application is that it requires several pieces of pre-shared information such as the recipient public address and smart contract address, as well as an agreement on a secret key before transaction through an off-chain channel.

Wang et al. [37] explored data storage and sharing issues in decentralized storage systems, and proposed a solution which utilizes IPFS, Ethereum blockchain, smart contracts, and attribute-based encryption (ABE) technologies. ABE is a type of PKC which can define fine-grained access control by using different attributes on deriving keys for the decryption process. ABE schemes conventionally use Private Key Generators (PKG) which act as a trusted party. For the sake of decentralization, the authors set up a system to distribute secret keys, and eliminate the trusted PKG. Smart contracts were used to enable keyword search function on the ciphertext of the decentralized systems. In the proposed framework, data owner and data user derive a shared key using Diffie-Hellman key exchange scheme, and this shared key is used for encrypting/decrypting the transaction data where secret key is stored. However, the channel for Diffie-Hellman key exchange is not described.

Using blockchain technology is also particularly beneficial for publishing digital content with author royalty and protecting intellectual properties of the creators. When the author saves her digital asset in the blockchain with a transaction she signed with her private key, it will be recorded with a timestamp in the blockchain. This means the blockchain acts as a notary and provides authenticity of the content, such as books, music, movie, etc. Most of the time, saving the hash of the digital content is a suitable way so that it would be cheaper to store, and authenticity of the content is still provable. Digital content created with collaboration, such as scientific research papers are also suitable for publication in the blockchain. In [38], authors implemented and evaluated a use case of this using Ethereum blockchain, smart contracts, and IPFS.

2.2. Fair Exchange of Digital Content

Fair exchange is a well-studied subject in distributed computations. A fair exchange protocol enables two actors to exchange items in a way that, either each player gets other's item or neither of them does [39]. Francez [40] identified several fairness notions in distributed systems with concurrency. Asokan et al. [41] described a generic protocol for fair exchange of electronic goods, such as confidential data, public data, or payment information. In his thesis, Asokan [7] adopted fairness to the protocols in electronic commerce where items of value are exchanged. In this context, an exchange is fair only if at the end of the exchange, either both parties receive the value they expect, or neither party obtains any extra useful information [7].

In order to provide fair exchange, many conventional systems and protocol propositions rely on TTPs. As a matter of fact, Pagnia and Gärtner [42] showed that it is impossible for a fair exchange protocol to completely assure strong fairness without a TTP. A TTP can be utilized either online or offline depending on the activeness of the TTP in the protocol. An online TTP is always active during the exchange, whereas offline TTP is active only in the case of fraud [43]. Thus, online TTPs bring overhead to the communication and raise privacy issues.

Protocols that assume parties would act honestly most of the time, and has an offline TTP which becomes active by a call triggered by one of the parties in case of a dispute are called optimistic fair exchange protocols [41] [44] [45] [39] [46] [47]. Even though

TTP is not used in optimistic protocols as long as both parties do not misbehave, relying on a TTP still has its drawbacks; such as honesty, reliability, and privacy issues. Distributing the trust to multiple arbiters is studied as a proposition to mitigate these drawbacks [46] [48]. Escrow services, which act as the TTP, are proposed as an arbiter in a peer-to-peer file sharing system with an optimistic protocol for efficiency [47].

Using blockchain and cryptocurrencies, it is shown that fair exchange can be achieved in a trustless manner [49]. Advance of the blockchain technology changed existing fair exchange schemes, and instead of a conventional TTP or escrow, it is possible to utilize secure and decentralized escrow protocols as a trustless third party [50] which releases money when the digital content is delivered. Smart contracts can take the role of the TTPs [51] to achieve fair exchange between parties that do not trust each other, similar to optimistic fair exchange protocols, but without suffering its drawbacks.

Enforcing monetary penalties to malicious users or mediators [47] [50] is proposed as an approach for fair exchange protocols. Likewise, there are protocols that enforce parties to deposit predefined amount of money as an incentive to achieve fair exchange. [52] [53] [54] [55]. Zero knowledge cryptography, i.e., zk-SNARKs can be also used to achieve fair exchange of sold goods in Bitcoin network, by utilizing Zero Knowledge Contingent Payment protocols [49]. Nevertheless, this is a cryptographically expensive approach and proposed due to the limitations of Bitcoin scripting language. Ethereum smart contracts can be used as the trustless mediator as proposed in [51].

CHAPTER 3

BACKGROUND

Since the internet and social media era took over, people started to realize the potential risk of centralization through power groups like nation states and giant internet companies and how these organizations, if corrupted, can manipulate financial instruments and digital data obtained from the people. Although the demand for decentralization of authority got stronger, due to technical limitations, it was not fully realized until decentralized technologies like blockchain emerged. Blockchain first started to use as the transaction ledger of the Bitcoin network and made decentralization really feasible for business.

3.1. Blockchain

Blockchain is a distributed ledger made from a list of records which are called blocks linked using cryptography in a peer-to-peer network. The idea of a cryptographically secure chain of blocks was introduced by cryptographers Stuart Haber and W. Scott Stornetta in 1991, as an attempt to create an immutable document timestamp records using cryptographic hash functions [56]. They even improved the efficiency and reliability of their digital timestamping service utilizing Merkle trees allowing multiple documents to be stored in a block [57]. Nevertheless, the blockchain as we know it was first introduced by Satoshi Nakamoto as the immutable transaction ledger of Bitcoin, the first and most famous cryptocurrency. Although Nakamoto, whose true identity is still unknown, did use block and chain terms separately, blockchain term got popular later on.

Despite the fact that *digital currency* term is not new, Bitcoin was the first deregulated and decentralized *cryptocurrency* in the market. In his/her famous paper [1] Nakamoto came up with this decentralized and peer-to-peer digital currency which can make possible to directly send and receive digital cash from one party to another without the need for any financial organizations in the middle. Proposing a solution to the double-spending problem using peer-to-peer networks, hence eliminating the need for a TTP was the remarkable innovation in the digital cash systems. This could be achieved by timestamping the transactions and hashing the transaction blocks in to a chain utilizing proof-of-work and as a result producing an immutable ledger.

Blockchain provides a distributed ledger which is cryptographically secure and runs on a peer-to-peer network. It is append-only, and updatable only with the consensus among peers, which makes it immutable (extremely hard to change, and cannot be changed without notice). Parallel to this characteristics, and Zheng et al.'s work [58], we can list the key features of the blockchain as follows:

- *Decentralization*: there is no third party, authority is reached with consensus among peers.

- *Immutability*: once the transactions are recorded in the ledger, the ledger cannot be changed without notice.

- *Anonymity*: transactions can be made with easily generated public addresses and asymmetric cryptography. The identity can be found using other internet means, but in the blockchain there is only addresses and keys, hence no personal information is stored.

- *Auditability*: all the peers can have the blockchain data so it is publicly available, and all the transactions can be traced back to the first block, aka genesis block.

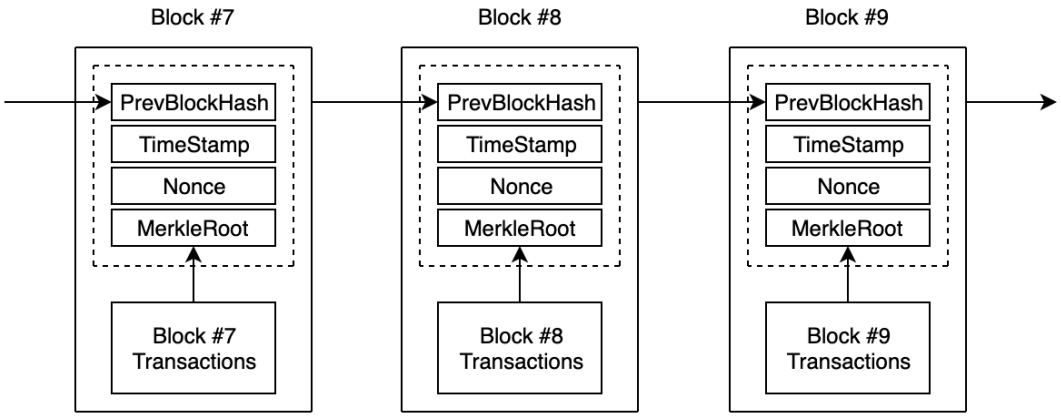

Figure 3.1: Simplified Blockchain

As a typical blockchain implementation, a Bitcoin transaction is executed as follows. Alice and Bob both have a pair of public and private keys, both users are uniquely identified in the network through their public addresses and PKC. Alice initiates a transaction to Bob over the blockchain network. This transaction message is broadcasted to the network participants, which are called nodes. These unconfirmed transactions stay in a pool called the transaction pool until they are verified and validated. These transactions are validated by miners and recorded in the public ledger with a process called mining. The miners create a block with the transactions they validated, and Merkle root of the hashes of these transactions is stored in the block header. For the miner to add a block to the blockchain she needs to solve a complex mathematical problem with the hash of the head of that particular block, which is called the proof-of-work. After a certain number of nodes reach a consensus, the new

block which contains these new confirmed transactions is immutably chained to the blockchain with cryptographic hash functions and the miner is rewarded as an incentive. Figure 3.1 illustrates a segment of a blockchain, and how the blocks are chained to each other in a simplified way.

Merkle root hashes are the root node of Merkle trees, which are a fundamental part of the blockchains for immutability assurance. Theoretically, it is possible to make a blockchain without Merkle trees, however this would yield scalability issues [59]. This is because Merkle trees are used to derive a single hash from a chunk of transactions to store in the block, instead of storing all the hashes of each transaction in the block separately. As depicted in Figure 3.2 [1], Merkle trees are sort of a binary tree, in which all the leaf nodes are hashes of transactions and other nodes are hashes of previous hash pairs. As a result of this architecture, even if a single transaction data changes, the root hash changes, which assures the immutability of all the transactions in the blocks with a single hash.

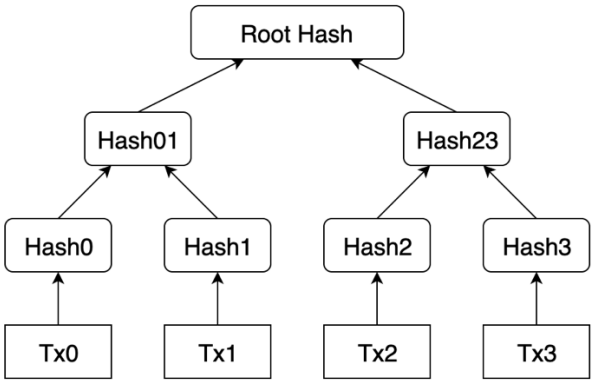

Figure 3.2: Transactions Hashed in a Merkle Tree

Blockchain technology gained enormous recognition with the introduction of Bitcoin, but in 2013 when Vitalik Buterin [2] came up with the idea of implementing smart contracts in a blockchain network he named Ethereum, the era of second generation blockchain technology has started by presenting and offering far more opportunities with the possibility to change the Web itself. Ethereum platform pioneers the next generation of decentralized applications in a discretionary distributed ecosystem as an attempt to achieve a whole new Web architecture.

In addition to Bitcoin and Ethereum, there exist numerous other blockchain networks which use other approaches to achieve basic characteristics of the blockchain technology and even change some of them, or offer even more features. For example Zcash is a digital currency concentrated on protecting the privacy of its users, which uses zk-SNARKS, a form of zero knowledge cryptography to achieve this [60]. Monero claims to provides secure, private and untraceable blockchain experience [61]. There are also several other blockchain platforms supporting smart contracts and offering infrastructure for decentralized applications, such as EOS and Quorum.

Blockchain technology provides a transparent and verifiable architecture that has the potential to alter the traditional way of exchanging cash and assets, enforcing contracts, and sharing data and information. Financial applications with a peer-to-peer cash payment system are the first adaptations of the blockchain technology in the real world, which is natural because of the way the system provided a new approach to currencies and digital currencies as cryptocurrencies like Bitcoin and Ether. Later with the support of scripts, more complex applications like value and property exchange, and smart contracts have been developed. With recent advancements in the ecosystems such as Ethereum platform, blockchain applications in the areas like government, health, science, industry, IoT, notary, data storage and many more have been actualized. Swan [62] described these three stages as Blockchain 1.0, Blockchain 2.0 and Blockchain 3.0 respectively.

There are also cloud providers such as Amazon, IBM, Microsoft and Google that provides cloud services for blockchain technology, i.e. blockchain-as-a-service (BaaS). For example an Ethereum modification JP Morgan's Quorum, R3 Corda, and Hyperledger technologies are available in Azure cloud provider [63]. Amazon Web Services (AWS) supports Ethereum and Hyperledger Fabric blockchains [64]. Google offers a click-to-deploy Ethereum service [65].

A few key points of blockchain technology, which inherently applies our further study in this thesis, are described as follows.

3.1.1. Consensus

In blockchain, a new block is permanently added to the blockchain using a consensus protocol, which ensures all the distributed nodes synchronize and store the same order of transactions and blocks, consequently provides integrity of the blockchain. Since the immutability and auditability is ensured by consensus algorithms, the security of the consensus model is very essential for the security of the blockchain. These algorithms have to be resilient to node failures and other possible network problems due to nature of distributed networks, and especially they have to deal with selfish and malicious nodes [66]. Wang et al. [67] highlighted the necessity of incentive mechanisms in permissionless blockchain networks as a common characteristic. There are several consensus algorithms proposed for different kinds of blockchains, but we are going to mention two major permissionless and distributed consensus algorithms for permissionless blockchains.

Bitcoin uses the consensus algorithm which Nakamoto [1] called Proof-of-Work (PoW). The nodes who validate transactions in the blockchain network are called miners. These miners solve a very hard mathematical problem with their CPUs or GPUs. The miner who solves the problem shares the solution with other mining nodes and these nodes confirm this solution and add the new block to their own copy of the blockchain and eventually the longest chain is accepted as the blockchain. The miner whose solution is accepted earns Bitcoin as a reward. Ethereum is also using PoW as its consensus mechanism.

The concept of Proof-of-Stake (PoS) is proposed as a modified PoW scheme in order to increase the scalability and reduce the huge electricity consumption of processing units due to exhaustive hash queries [67]. In PoS, nodes cannot freely join validation process, instead only nodes that made a significant security deposit, which is called stake, can validate the transactions in Ethereum blockchain [66]. Ethereum is planning to switch to a variant of PoS scheme called Casper with its major Serenity upgrade. In Casper, the stake holders have the vote right according to their stakes. Different variations of PoS scheme is applied in several other blockchain networks.

3.1.2. Decentralization

Decentralization of authority is one of the most important aspects of blockchain technology. Strong decentralization eliminates the need for trust to any kind of central authority in the network without compromising the trustworthiness of the system [68]. Even though all the nodes are trusted equally in an ideal blockchain system to maximize decentralization, it is not always easy to achieve satisfactory decentralization while designing blockchain systems. Due to possible 51% vulnerability (see Section 3.1.5) in PoW consensus based blockchain systems, satisfactory level of decentralization is crucial for a blockchain.

In the Nakamoto PoW mechanism, nodes spend computing resources in order to earn rewards. Since the chance of getting the reward increases with the amount of the computational power of the miner, in Bitcoin network, power has been significantly biased towards a few nodes which are called mining pools [69]. As a result of this, generally in PoW scheme, risk of centralization increases with the growth of the blockchain network [70]. This issue is often referred as democratizing the mining power. For example, several blockchains utilize ASIC-proof hashing algorithms, as these processors are expensive, thus they affect the decentralization of the blockchain negatively. Kwon et al. [69] addressed the difficulty of designing a good decentralization and proved that a good decentralization is not possible if the blockchain system has no Sybil costs, which actually means if the gap between rich and poor in the real world is not reduced, level of decentralization will not be able to high forever with a high probability.

3.1.3. Benefits

The reason that made blockchain technology so in demand is the benefits the technology offers. Several advantages and benefits of blockchain technology for financial institutions as well as in several other industries are discussed in many studies [4] [71]. A comparison to conventional cash transfer systems would depict the potential of the technology: with blockchain, someone has transferred $1.2 billion worth cash in 1.1 seconds with a cost of $0.015 [72], which would take days [73], even weeks with a significant amount of cost if the cash was transferred through banks.

We can list the main benefits of the blockchain technology derived from its natural characteristics as follows: *Decentralization* of the authority, hence no need for a TTP as the system is trustless by itself. *Reliability* and *availability* because of its peer-to-

peer network with no central point of failure. *Security* as all the transactions and the blockchain itself are secured with cryptographic hash functions and PKC. *Immutability* as once the data is added to the blockchain, it is extremely hard to change it without noticing. *Smart contracts* which enables to change many of the current paradigms in business with its autonomous and trustless characteristics. There are other potential benefits of the blockchain technology as published in [74]. Nevertheless, it should be noted that, most of the time it is hard to achieve all the benefits at the same time as some of the benefits can contradict each other and in some cases the benefits are dependent to each other.

3.1.4. Limitations and Scalability

Although blockchain technology has gained enormous attention from both academia and industry, the concept is relatively very new. The technology is still immature, lacks regulations, and has adaptability problems which limits its usability in different areas of education and business. In traditional PoW consensus schemes, miners have to download all the chain data, which is more than 200GB for Bitcoin [75] and more than 100GB for Ethereum [76]. When the size of the blockchain data dramatically increases in the future, mining process will not scale well with current mining mechanisms of Bitcoin and Ethereum.

Distributed and decentralized solutions like blockchains are inefficient compared to the traditional centralized database solutions and client-server architecture, which means decentralization is achieved with the cost of scalability limitations. For example with a generalized state transition function of Ethereum, it is very hard to partition and parallelize transactions to apply a divide and conquer scheme [77]. Hence, it is more difficult to scale up to higher capacity, and because of its design it is harder to make changes, which results in less flexibility for the blockchain [74].

A fully decentralized blockchain limits scalability because of the throughput upper bound [68]. As a result of this, it prevents scaling of the smart contract execution too, as it was experienced when a decentralized application named Cryptokitties crippled the Ethereum blockchain in late 2017. Current permissionless blockchains process 3-20 transactions per second which is way less than mainstream payment systems [22]. It follows that, blockchain systems are faster than centralized systems in validating transactions due to its nature of network and validation mechanisms, compared to transactions per second capacity of traditional centralized networks like Visa and PayPal. However, Bitcoin and Ethereum network are still very slow because of this aforementioned scalability issues.

There are several studies that propose improvements to scalability problem of blockchains, naturally the communities behind Bitcoin and Ethereum are also trying to solve this problem too. Ethereum community is planning to solve the scalability problem of Ethereum blockchain using sharding [68]. Via sharding, nodes store and process only the data that is relevant to them which decreases the amount of resources required to join the network [78]. Sharding in this context simply means fragmenting the network in to pieces without compromising Ethereum network's decentralization and security aspects. Currently Ethereum has 15 transactions per second capacity and

with sharding it is anticipated that it will be at least 1000 transaction per second and even more.

3.1.5. Potential Vulnerabilities and Risks

Even though blockchain technology provides a reliable and cryptographically secure architecture, there are still some potential vulnerabilities and risks needs to be discussed. Some of these vulnerabilities and risks are inherited from its online nature similar to other online systems, and some of the vulnerabilities are specific to genuine design of the blockchain technology itself. In this section we address permissionless blockchain mechanism related vulnerabilities, such as 51% vulnerability, private key security, criminal activity, double spending, transaction privacy leakage and Sybil attacks [79] [80] [81]. Smart contract specific vulnerabilities and related security issues are discussed in Section 3.3.3.

- *51% Vulnerability*. As a result of the PoW consensus mechanism, when a miner or mining pool gains majority of the hashing power, the malicious miner can launch a 51% attack to take over the validation and verification process of transactions and control the entire blockchain. With this majority of computational power, attacker can modify/reverse the transaction data which may result with double spending, modify the ordering of transactions, stop the block verifying process of normal transactions, and stop other miners mining any available block [79] [81].

- *Private key security*. In blockchain, user's public address is her identity and related private key is her security credential to access her account. Since there is not any TTP in the architecture, if the private key is lost or stolen, it is almost always impossible to get the account back. Blockchain accounts are generally handled with account managers acting as a vault, which ensures the security of the private keys. Software wallets, i.e., hot wallets, are nothing but a standalone or web applications, thus they are intrinsically exposed to hacks.

- *Criminal activity*. Most of the time it is hard to track the identity of a blockchain account in permissionless blockchain systems. There are even zero proof knowledge blockchains that secures the privacy of the account. Because of this the system has the risk to be used for illegal activities in the internet and cause bad reputation, which was the case for Bitcoin until the technology was very well recognized recently.

- *Double spending*. PoW based blockchains tries to prevent double spending with time stamping the transactions in a block and broadcasting it to the all network. Because of the nature of the consensus mechanism, there is a chance that an attacker leverage race attack for double spending during the intermediate time between the transactions' initiation and confirmation, and before the second transaction verified to be invalid, the attacker gets the first transaction's output, which causes a double spending [79]. In order to avoid this race attacks, many merchants wait for at least 6 confirmations of a transaction before considering it as successfully finalized.

- *Transaction privacy leakage*. Since blockchain is an open architecture, all the transactions can be traced back to genesis block. Although users have public addresses as their identity, analyzing of blockchain data can reveal some private information. In an attempt to solve this issue, blockchain platforms Zcash uses zero proof knowledge mechanism to secure the privacy of its users. Lack of privacy of the transaction data is one of the major barriers for mainstream adaptation of blockchain technology by the industry.

- *Sybil attacks*. Due to nature of the permissionless blockchain, a single malicious user can generate multiple fake identities all controlled by this single user, which appear to be unique benign users to the rest of the network. Sybil attack occurs when the attacker uses these fake accounts to influence and manipulate the network with its voting power during the consensus process. Since this attack increases the reputation of these malicious pool, other users might join this attacker's mining pool as well in order to earn more reward without knowing its true intentions. Increasing the cost to create an identity is a preventive measure, and PoW mechanism implement this by requiring expensive hardware to join and mine in the network. Another mechanism proposed to mitigate Sybil attacks is trust and reputation mechanisms [82] as implemented in TrustChain [80].

3.1.6. Blockchain Types

There are three types of blockchain networks regarding how they accept nodes for the network consensus:

- **Permissionless (Public) Blockchain**. This blockchain network and its data is publicly accessible, and anyone can participate in the consensus process anonymously. Typically, there is a cryptocurrency of the system used for incentivizing members to join the mining process. Ethereum and Bitcoin are examples of permissionless blockchain networks.

- **Permissioned (Private) Blockchain**. This blockchain network typically belongs to a company. There is an access control layer trough which network owner determines who can join the network and even specify who can verify the transactions, which as a result causes the centralization of the authority on the blockchain network. This central authority can also revoke permissions for the users that can access the system or participate in the validation process.

- **Consortium (Public Permissioned) Blockchain**. This blockchain network can be considered as a hybrid of private and public blockchains. The consensus process is controlled by a group of preselected nodes, typically a consortium of companies, and no other parties can join or interfere with decision making. Hyperledger and Quorum (Ethereum based) are examples of projects to build permissioned blockchain applications.

Each blockchain network has its own advantages and disadvantages. According to Wust and Gervais, there are use cases suitable for each of the technologies [5]. Zheng

et al [83] explored the blockchain types and made a comparison of these blockchain networks which is given in Table 3.1. We added a transaction cost row in the table as an addition. Since permissionless networks require consensus schemes such as PoW, transactions costs are higher than permissioned blockchains.

Table 3.1: Comparison of blockchain networks

Property/Type	Public	Consortium	Private
Consensus	All miners	Preselected set of nodes	Single entity
Read permission	Public	Public/restricted	Public/restricted
Immutability	Yes	Could be tampered	Could be tampered
Efficiency	Low	High	High
Transaction cost	High	Low	Low
Decentralized	Yes	Partial	No

3.2. Ethereum

Ethereum is an open source, globally decentralized computing infrastructure which executes smart contracts written in a Turing-complete programming language [84]. Vitalik Buterin [2] defined Ethereum as a blockchain with a built-in Turing-complete programming language, that allows anyone to write smart contracts and decentralized applications; with the intention to have a platform that allows developers to create consensus based applications with scalability, standardization, feature-completeness, ease of development, and interoperability.

Similar to Bitcoin, Ethereum platform has its own digital currency, named Ether, which is necessary in the operating of the blockchain as a transaction fee for metering and constraining execution resource costs [84]. Due to their decentralized blockchain architecture, there is no one controls or owns Ethereum and Bitcoin networks. Nevertheless, Ethereum offers more than just a distributed digital cash payment network, thanks to its smart contracts running in a global virtual machine shared across all the network. This virtual machine running in every node is called Ethereum Virtual Machine (EVM), and as a whole, Ethereum network acts as “the world computer”.

EVM is a global virtual machine distributed among the nodes of its peer-to-peer network. Ethereum smart contracts, which are basically just computer programs, run on this EVM. These programs can be written with a user-friendly programming language, such as Solidity and Vyper. EVM is an isolated runtime environment running on nodes and because of its sandboxed nature smart contracts running on EVM have no access to the resources of the node hosting it. In Ethereum, transactions are made in order to: create new contacts, call functions of a contract, and transfer Ether to contract accounts or external accounts [85]. The transactions that change the state of an account requires computation, and every computation on the network is actually done by nodes delegated with EVM. Since these computations are handled with real

hardware resources on nodes, there is a price for every computation occurred as a result of a transaction. This price is called Gas. Transaction cost is important to protect the network from intentional or unintentional malicious and heavy computational tasks like DDoS attacks.

As described above, each node in the Ethereum network runs the EVM and all the nodes executes the same instructions. This massive parallelization of the computing makes computation on Ethereum slower and more expensive than on a traditional computer. But the goal of this process is to maintain consensus across blockchain, in an attempt to achieve extreme levels of fault tolerance, zero downtime, unchangeable and censorship-resistant data storage on blockchain [86]. Transactions costs and consensus rewards are collected by miners which provides the economic incentive for miners to dedicate hardware resources and electricity to the Ethereum.

An Ethereum network can be created trivially by creating a genesis block, i.e., the first block of an Ethereum blockchain. Actually, there are many Ethereum blockchains operating on the internet. The public Ethereum that we know, runs on its permissionless main network, called Ethereum Mainnet. There are other public Ethereum networks created for smart contracts testing purposes, such as Ropsten, Kovan, and Rinkeby. Ethereum can also be deployed as a private blockchain, in a permissioned network, in a company or in a consortium. As a matter of fact, an Ethereum blockchain can be run on a single computer, as well, mostly for smart contract development purposes.

3.2.1. Accounts

An account in Ethereum is an object that stores the state of an ordinary user or smart contract, and it is uniquely identified with a 160-bit address. An Ethereum account contains four fields [2]: Nonce, balance, contract code (if present), and a key-value store called storage. The state of the Ethereum network is actually the state of all the accounts, which is updated with every new block.

There are two types of accounts [2] [86]:

- *Externally owned accounts (EOAs)*, which are controlled by private keys, and do not have associated code.

- *Contract accounts*, which are controlled by their contract code, and have associated code.

Externally owned accounts represent identities of external agents such as human users and mining nodes. Users need these accounts to interact with Ethereum blockchain by sending messages via transactions that are signed by accounts' private keys. An externally owned account can send messages to other externally owned accounts, or to other contract accounts. A message between two externally owned accounts are basically a value transfer, like a Bitcoin transaction. A message from an externally owned account to a contract account activates the code of the contract account, allowing it to read and write to internal storage and send other messages or create

contracts in turn [2]. A message from an externally owned account to a contract account has a data field addition to the value field. Contract accounts cannot initiate new transactions on their own, like mentioned above, they can only fire transactions as a response to other transactions they have received. Therefore, all the action that occurs on the Ethereum blockchain is always set in motion by transactions instantiated from externally controlled accounts [86].

3.2.2. Messages, Transactions, and Calls

Messages in Ethereum platform are similar to transactions in Bitcoin but there are three important differences [2]: an Ethereum message can be created by a user or a smart contract, Ethereum messages can contain data along with value, and if the recipient of the Ethereum message is a contract account it can return a response. Components of a message is given in Table 3.2 [2][86].

Table 3.2: Message Components

Message
Sender Address
Recipient Address
VALUE
Data (optional)
STARTGAS

In Ethereum, transaction is cryptographically signed data package that stores a message originated from an externally owned account and sent to another externally owned account or contract account on the blockchain. Components of a transaction is given in Table 3.3 [2][86].

Table 3.3: Transaction Components

Transaction
Recipient Address
VALUE
Data (optional)
STARTGAS
GASPRICE
Signature

There are two types of transactions: message calls and contract creations. Essentially, a message is like a transaction, except it is produced by a contract and not an external actor, and it is not signed, thus it is not explicitly included in the blockchain. An important consequence of the message mechanism is the idea that contracts accounts

have equivalent powers to externally owned accounts, including the ability to send message and create other contracts [2] [86]. Transactions changes the state of the blockchain, and they are used for writing data to the network.

In order to read data from the network, calls are used. By using calls, code can be executed in the network, but it does not change a data in the blockchain. A function in a smart contract can be executed by a call, and it returns a value immediately; whereas transactions need to be mined, thus require some time to finalize. Differences between transactions and calls are given in Table 3.4 [87].

Table 3.4: Transactions vs Calls

Transactions	**Calls**
Write data to the blockchain	Read data from the blockchain
Cost gas for the execution	Free to run
Change the state of the network	Do not change the state of the network
Wait for mining	Processed immediately
No return value, but a result object	Return value

3.2.3. Gas

When the EVM executes smart contracts, it accounts for every instruction (computation, data access, etc.) with a predetermined cost in units of gas [84]. For example, reading data is more expensive than basic mathematic calculations, and writing data is even more expensive than reading data. Gas costs for several common Ethereum operations in the EVM is given in Table 3.5 [86]. As can be seen, storing large amount of data in blockchain is not a good practice. Gas is one of the most crucial components of Ethereum operating architecture. To begin with, it serves as the main economic incentive in Ethereum network, and also it provides protection against DoS attacks and faulty smart contracts that could drain the resources of the network. The price per unit of Gas is represented in Gwei, and 1 Ether is equal to 10^9 Gwei, and 10^{18} Wei.

There are two main components of the Gas mechanism: Gas Limit and Gas Price. Gas Limit is the maximum number of Gas a user is willing to spend on a computation. Gas Price is the amount of Wei a user is willing to spend on every unit of gas. Users can increase or decrease the Gas Price for their transactions to have their transactions mined faster or slower, respectively. This happens because high Gas Price attracts more miners, and if the Gas Price is not sufficient the user might wait for a long time for her transactions to be mined.

Table 3.5: Operation Costs in EVM

Operation Name	Gas Cost	Remark
step	1	default amount per execution cycle
stop	0	free
suicide	0	free, refund given
sha3	20	
sload	20	get from permanent storage
sstore	100	put into permanent storage
balance	20	
create	100	contract creation
call	20	initiating a read-only call
memory	1	every additional word when expanding memory
txdata	5	every byte of data or code for a transaction
transaction	500	base fee transaction changed
contract creation	53000	changed in homestead from 21000

The initial cost of a transaction is the product of Gas Limit and Gas Price [2] [86]. With smart contracts a user cannot know in advance the exact amount of Gas that will be used for the related computations in the EVM. In Ethereum, transaction costs are paid in advance, so the user have to specify the Gas Limit during the initiation of the transaction. When the smart contract execution successfully ends, unused gas is refunded to the source address. If the amount of gas used by the smart contract exceeds the Gas Limit, execution of the smart contract throws and exception and all the modifications done in contract execution are reverted to the original state. In this case, the amount of Gas spent is not refunded to the source address. For this reason, it is a good practice to set the gas limit reasonably high. Given the above, we can find the overall cost of a transaction after it is finalized as follows.

$$Overall\ transaction\ cost = (Used\ Gas)\ x\ (Gas\ Price)$$

3.3. Smart Contracts

In Ethereum ecosystem, a smart contract is an autonomous computer program that runs in EVM. It has a set of predefined functions and logic which it executes accordingly. Smart contracts are immutable computer programs, which means once the smart code is created and deployed in the blockchain; its state can change, but its code, thus its predetermined behavior cannot be changed. Smart contracts are Ethereum accounts which store their associated code in the blockchain, and they are uniquely identified by their 160-bit addresses. With the special data field of contract account, smart contracts can store data which can be used to store information, balances, or any other data necessary for implementing the application logic. As a result of the decentralized characteristic of its underlying blockchain architecture, smart contracts enable

exchange of cash, property, digital asset, information, and value without the need for a trusted middleman.

There are other implementations of smart contracts in other blockchains, too. For example, Bitcoin provides a Turing-incomplete scripting language to create custom limited smart contracts [88]. Bitcoin's scripting language supports a large subset of computations but main functionality it is missing is loops [2]. On the other hand, Ethereum implements Turing-complete EVM code, including loops. This makes Ethereum an outstanding smart contract framework.

Smart contracts are written in a high-level programming language and compiled to low-level bytecode that runs in the EVM. When compiled, they are deployed to the Ethereum blockchain by a special *contract creation* transaction. Contracts only run if they are called by a transaction originated by an EOA, either directly or over another contract call. Contracts can call other contracts, hence that contract's functions, via message calls. Although a contract's code is immutable, a contract can be deleted by removing its code and internal state, which leaves an address of an empty account. In order to be able to delete a contract, the contract must be programmed with that functionality from the beginning. When a contract is deleted, some gas is refunded to the contract owner. This incentivizes the users to delete unused contracts, and thus release the network resources [84].

The most popular and adapted application for smart contracts is custom tokens. Tokens are basically exchangeable cryptocurrencies, specific to a decentralized application (DApp). A video arcade is a simple analogy to understand why tokens might be necessary when we already have Ether as a payment method [89]. There are hundreds [90] of custom Ethereum tokens in use, with different exchange rates, and serving for several purposes in particular applications. Tokens are mostly used as utility tokens or equity tokens [84]. Antonopoulos and Wood [84] define utility tokens as the tokens required to gain access to a service, application, or resource, and equity tokens as the tokens that represent shares in control or ownership of an application or startup. There are two token standards in Ethereum: ERC20 and ERC721 Token Standards which define fungible and non-fungible tokens, respectively.

Token sale is a popular way to raise funds for Ethereum projects. This crowdsourcing is called Initial Coin Offering (ICO) and, there are thousands [91] of Ethereum projects have been trying to finance their blockchain project with ICOs. Most of the ICOs is managed through ERC-20 Token Standard Contract [92]. Nevertheless, since there is no regulation in ICOs there are also many scam projects as well, which requires special analysis for decision making [93].

Related to equity tokens, another application of smart contracts worth to note is Decentralized Autonomous Organizations (DAOs). DAOs are complex, long-term smart contracts that encode the bylaws of a decentralized organization [2]. Using smart contracts, ownership in an organization and terms for the disbursal of funds can be specified at the outset. The smart contract can be written such that it can be only changed with the approval of the majority of the owners [86].

Solidity and Vyper are two notable high-level programming languages designed for developing Ethereum smart contracts that run in the EVM. These programming languages and security issues in smart contracts are described further in the following sections. Note that, security issues related to blockchain architecture of Ethereum is already addressed in Section 3.1.5.

3.3.1. Solidity

Solidity is the primary smart contract programming language that runs on the EVM. It is an object-oriented, high-level language, with a syntax similar to JavaScript, and influenced by C++ and Python, as well. It is a statically typed language, and supports inheritance, libraries, and complex user defined types [94]. Solidity smart contract source file is composed of contract definitions, import directives, pragma directives, and comments. Contracts in Solidity are like classes in object-oriented programming languages, and likewise, contracts can inherit from other contracts. Each contract can contain declarations of State Variables, Functions, Function Modifiers, Events, Struct Types and Enum Types [94].

Solidity code is compiled by Solidity compiler in to low level bytecode that can be executed by the EVM. This bytecode is deployed to an Ethereum network, such as Mainnet, Ropsten, Rinkeby, etc., in that particular contract account's code storage. Bytecode can only be understood by EVM, as a result of this, in order to access the smart contract's functionalities through DApps, Solidity compiler also emits an Application Binary Interface (ABI). ABI is a JSON format description for the smart contracts that makes it possible to understand and access functions of a contract, and read/write data of it. When the contract is deployed, a contract address is created. Using this contract address and its ABI, a contract instance can be accessed and used from a decentralized application over JavaScript calls. These processes are illustrated in Figure 3.3

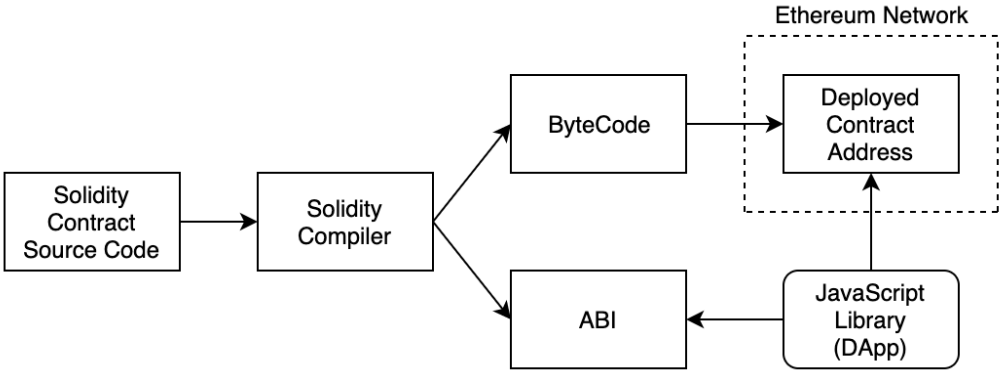

Figure 3.3: Solidity Bytecode and ABI

3.3.2. Vyper

Vyper is a contract-oriented, Python like smart contract development language for the EVM. Contracts in Vyper are similar to classes in object-oriented programming

languages, and each contract can contain declarations of State Variables, Functions, Events, structure-struct types and contract interfaces [95].

A study in 2018 [96] analyzed one million deployed Ethereum smart contracts, written in Solidity, according to a systematic characterization of a class of trace vulnerabilities they presented. They focused on three vulnerabilities with properties such as smart contracts that either lock funds indefinitely, leak them carelessly to arbitrary addresses, or can be killed by anyone. These vulnerable smart contracts are named greedy contracts, prodigal contracts, and suicidal contracts, respectively. According to this study, they found out that many of the smart contracts have these trace vulnerabilities. Smart contract vulnerabilities are caused via bad practice of code. This is why, Vyper is designed with the purpose of making it easier to write secure code, or equally making it harder to accidentally write misleading or vulnerable code [84]. Vyper achieves this with principles and goals such as security, language and compiler simplicity, and auditability. Although Vyper enables developers to write relatively secure codes, the language itself is still experimental.

3.3.3. Smart Contract Security

Smart contract code, transactions, and all the data on the immutable Ethereum blockchain are public, and any state and its data can be accessed and read in a node's local copy without leaving a trace in the system. As a result of this, executing smart contracts in Ethereum brings certain risks. Because smart contracts deal with cash, these risks and issues often results in catastrophic situations.

The well-known DAO attack took place in 2016, in which hackers stole $50 million worth Ether, and induced a hard fork in the Ethereum blockchain. This attack was caused by the security flaws in the smart contracts introduced by Solidity language, although the DAO was working as intended [97]. In order to have a secure smart contract, code and contract should execute how developer intended, and should not lose ether.

There are several known smart contract vulnerabilities. Similar to Open Web Application Security Project's (OWASP) prominent list of top 10 application security risks, Decentralized Application Security Project (DASP) [98] has a list of top ten vulnerabilities for smart contracts. Atzei et al. [88] also investigated the attacks against smart contracts comprehensively, and listed 12 smart contract vulnerabilities, as listed in Table 3.6.

Defensive programming style is a good approach to avoid vulnerabilities for smart contracts, with the following best practices: minimalism/simplicity, code reuse, code quality, readability/auditability, and test coverage [84]. Since Ethereum ecosystem is still young and vibrant, Solidity language specifications change often, and being up-to-date is important, as well. During a complex developing process, the programmer should use open source and tested industry standard contract security patterns, such as OpenZeppelin [99] contracts. This reuse practice can be considered as parallel with reusing proven cryptographic libraries, instead of writing by one's own. The programmer also should use Ethereum smart contract security best practices provided

by Consensys [100], and original Solidity documentation [101], in order to avoid vulnerabilities and write secure smart contracts.

Table 3.6: Taxonomy of vulnerabilities in Ethereum smart contracts

Level	Vulnerability	Cause
Solidity	Call to the unknown	Invoking the fallback function
	Gasless send	Incurring in an out-of-gas exception
	Exception disorders	Irregularity in how exceptions are handled
	Type casts	Type-check error during contract execution
	Reentrancy	A function is re-entered before its termination
	Keeping secrets	A field is visible in the node even if it is private
EVM	Immutable bugs	A contract cannot be altered after it is deployed
	Ether lost in transfer	Ether is sent to an orphan address and lost
	Stack size limit	The call stack exceeds the limit of 1024 frames
Blockchain	Unpredictable state	State of the contract is changed before transaction
	Generating randomness	A malicious miner bias the seed for randomness
	Time constraints	A malicious miner choosing a suitable timestamp

Using security tools is a reasonable approach while writing complex, or cash-heavy smart contracts. In 2018, new security tools are introduced into the Ethereum market, which made it easier to build secure Ethereum smart contracts and DApps [102]. Remix, Trail of Bits, Securify, MythX, Manticore, Oyente are a few examples of these tools that performs static or dynamic analysis of Ethereum smart contracts.

3.4. Privacy

The user accounts can be considered as anonymous in permissionless blockchains such as Bitcoin and Ethereum, unless the user reveals her identity and public key. However, as all the transactions are readable; rather than the name of the user, users public address is recorded in transaction, thus, all transactions are actually pseudonymous [103]. It follows that, these pseudonyms can be deanonymized in this public scheme [21] [104]. For example, third-party web trackers can achieve this deanonymization if users use cryptocurrencies for the payments [105]. As can be seen, the anonymity and user privacy are hard to fully achieve in Ethereum.

In the context of the privacy of the blockchain, privacy can be generalized as the security of the blockchain data. This data is not limited with cryptocurrency transactions, but also smart contract binary code, and all the sensitive content of the blockchain, as well. Public blockchains such as Bitcoin and Ethereum are actually public records and since they have permissionless architectures, anybody can join the network, and access/read blockchain data. The privacy issue is a significant problem

with permissionless blockchains, and along with scalability, privacy is the main concern that hinders the mainstream usability of blockchain [22]. Typically, companies and individuals hesitate to use the technology because storing sensitive data on a place where everybody can read is generally not a good idea.

There are different strategies to overcome the privacy issue in permissionless blockchains. Cryptographically secure obfuscation, zero-knowledge proofs, using one-time accounts, encryption of the data are different approaches, which have their own advantages and disadvantages [22]. Ethereum community is aiming to solve the privacy issues in its next major update, by including zk-SNARKs in its on-chain verification process.

There are also several blockchain projects trying to address privacy issues of permissionless blockchains: Hawk project is a blockchain-based smart contract platform that stores encrypted transactions on the blockchain, and provides a platform to write private smart contracts [25]. Zcash utilizes zero knowledge proofs cryptography (i.e., zk-SNARKs) and one time accounts to provide security and privacy of the platform [23]. Lightstreams has developed a protocol called Permissioned Blocks and manage access to protected content in decentralized networks, integrating blockchain and distributed file sharing [106]. Enigma uses trusted execution environments, rather than zero-knowledge-proof, to allow nodes to make computations using encrypted fragments of the smart contracts without the need to decrypt them [26]. Enigma can act as a side-chain to Ethereum and provides users with the ability to write secret smart contracts.

3.5. Ethereum Web 3.0 Stack (Decentralized Web)

Web 2.0 is the name used to describe the era of the World Wide Web (WWW, Web) after 2000s, where the technology moved from static pages to interactive services like social media applications, which enabled people to contribute and share information online. Web 2.0 can be considered as the participatory Web [9]. Nevertheless, all these services and applications are controlled by a middleman, in a client-server architecture where all the data are stored in centralized databases.

Web 3.0 actually refers to the transition from client-server internet to a decentralized and secure internet, in an attempt to make it censor and control free. Web 3.0 replaces centrally owned applications with decentralized applications, client-server architecture with peer-to-peer network, the need for a trusted middleman with trustless blockchain consensus, centralized databases with distributed storages, and application logic with smart contracts. Note that these Web 2.0 and Web 3.0 definitions are not official denotations or technical specifications for the different aspects of Web, but a convention to differentiate the new applications and the technology behind it.

Ethereum has transformed itself from a mere platform of smart contracts and decentralized applications in to a large software ecosystem. Ethereum community has been developing more components to fully decentralize web applications and offers a full backend to a decentralized internet, aka Web 3.0 [86]. These concept and

technologies are still in early stages and most of the components are still in alpha versions. Ethereum offers an internet with built-in money and payment system, users owning their own data, and an open financial system and permissionless infrastructure controlled by no middleman.

Compared to current internet and web applications architecture, in Ethereum Web 3.0 stack: instead of servers, computations are done on EVM in a peer to peer network. Instead of web servers, hosting is done on distributed storages like Swarm and IPFS. Instead of HTTP API, service layer is handled by smart contracts. Instead of databases, storage is done on distributed storages or decentralized databases. Web 3.0 abstracted stack is depicted in Figure 3.4 [107]. We already explored Ethereum blockchain related concepts in earlier sections, other concepts in Ethereum Web 3.0 stack are elaborated in the following sections.

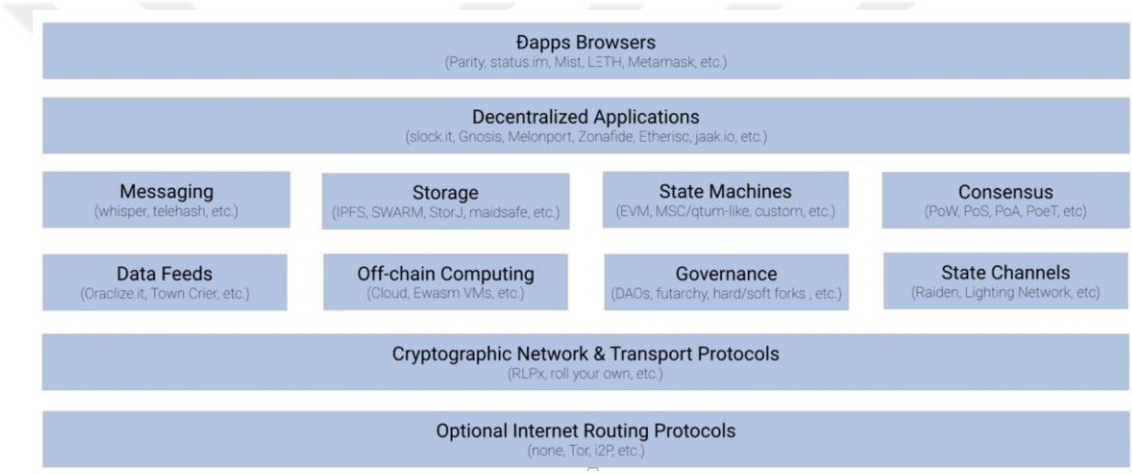

Figure 3.4: Web 3.0 Abstracted Stack

3.5.1. Decentralized Apps (DApps)

Decentralized applications are computer applications that run on a distributed network. In Ethereum, a decentralized application (DApp) refers to a web application that runs on the Ethereum network. Thousands of DApps have been developed on the Ethereum blockchain, in several categories like finance, games, insurance, etc. [108].

Application logic and payment functions of DApps are handled by Ethereum smart contracts as back-end software. In addition to smart contracts, DApps have a web front-end user interface as well, which is why a DApp does not look different from an ordinary web application to the end user. A simplified DApp architecture is illustrated in Figure 3.5.

The possible components of a DApp are as follows [84]:

- Back-end software (smart contracts)
- Front-end software (HTML, CSS, JavaScript, etc.)

- Data storage (Swarm, IPFS)
- Message communications (Whisper)
- Name resolution (ENS)

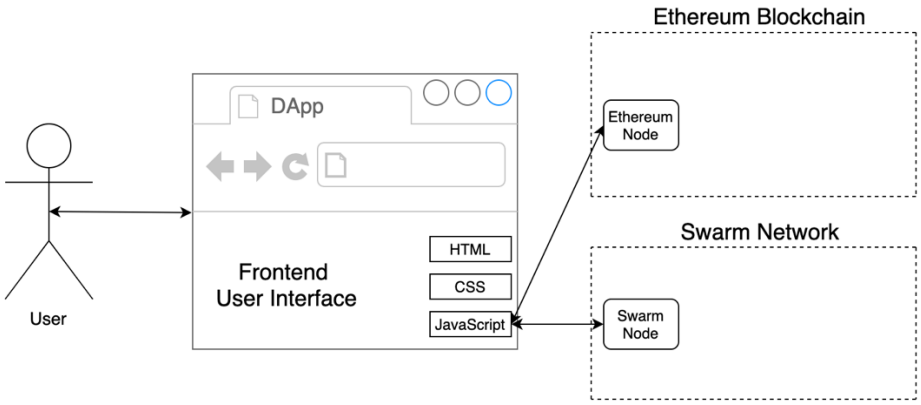

Figure 3.5: Simplified DApp architecture

Each of these components can be decentralized or left centralized. If all the components are decentralized, we can consider this DApp as entirely decentralized. Compared to conventional centralized web applications, DApps provides other advantages like resiliency, transparency and censorship resistance [84].

As the back-end software of a DApp, smart contracts store the program logic of the application, replacing the regular server-side web APIs. As we discussed in Section 3.2.3, smart contracts computations are very expensive, so a DApp developer need to be careful to keep the smart contract as minimal as possible, by identifying the aspects of the application that really needs to be decentralized on the Ethereum blockchain.

The front-end is linked to Ethereum via the web3.js JavaScript library. Web3.js is a collection of libraries which enables interaction with a local or remote Ethereum node, using an HTTP, WebSocket or IPC connection [109]. Using web3.js, you can retrieve user accounts, send transactions, interact with smart contracts, etc. As well as the Ethereum network, web3.js also enables interaction with Ethereum's peer-to-peer storage network called Swarm, and Ethereum's peer-to-peer messaging service called Whisper, through web3-eth, web3-bzz, and web3.shh APIs respectively [84].

3.5.2. DApp Browsers (Metamask)

While running a DApp in a web browser, users need a bridge to interact with Ethereum for signing messages, sending transactions, and managing keys. MetaMask is the most popular web browser extension that allows users to run Ethereum DApps in web browsers such as Chrome, Firefox, and Opera. It is an account manager and it includes a secure identity vault, providing a user interface to manage identities on different DApps ,and sign Ethereum transactions [110]. The typical user interface of MetaMask

is shown in Figure 3.6, over which a user can send and deposit Ether, and sign transactions.

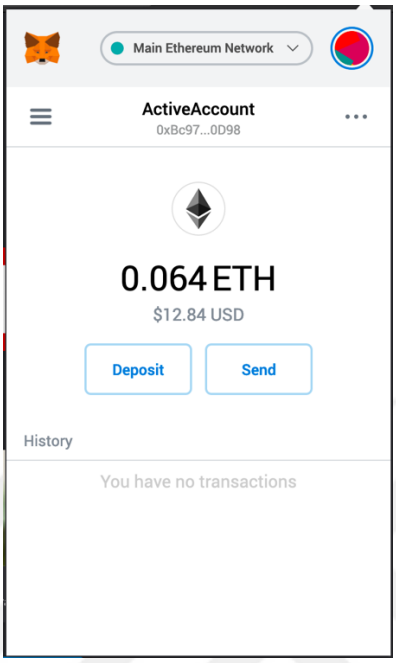

Figure 3.6: MetaMask User Interface

3.5.3. Decantralized Data Storage (Swarm)

Since storing and processing large amounts of data in smart contracts costs a lot of Gas, Ethereum blockchain is not suitable for storing large data, therefore most DApps require storing their data in a place somewhere outside of the blockchain. This off-chain (which is a term used to refer platforms off the blockchain) data storage can be centralized, such as a cloud storage service like Amazon S3 or a typical database. DApps also require a front-end interface for the users to interact with the smart contract easily. Likewise, this web interface data can be hosted in a centralized way, on a web server like a regular web page serving over HTTP and HTTPS protocols.

Although these centralized solutions provide a good working environment, they violate the decentralized principle of Web 3.0. They also become a possible target to DoS attacks, and suffering censorship. In order to overcome these problems and maintain the decentralization, distributed data storage platforms can be used for storing large static assets such as media files and static web interface files such as HTML, CSS, and JavaScript files.

Interplanetary File System (IPFS) and Swarm are the most common decentralized data storage and content publication platforms. IPFS is a content addressed, versioned, peer-to-peer file system [111] for storing and accessing files, websites, applications, data, and any digital assets. Thanks to its decentralized architecture, IPFS does not have a single point of failure, and its nodes do not need to trust each other [112]. Main idea behind IPFS is to replace and build a better decentralized web [113]. Similarly,

Swarm is a content addressable peer-to-peer storage system and content distribution service, a native base layer service of the Ethereum Web 3.0 stack. Providing a sufficiently decentralized and redundant service for storing and distributing DApp code, DApp data, and Ethereum blockchain data, is the primary objective of Swarm [114]. For the end user, as it is it with DApps from other traditional web applications, Swarm is not much different from the WWW. Thanks to its architecture, Swarm offers a decentralized web which is fault tolerant, censorship resistant, DDoS resistant, zero downtime, and self-sustaining [115].

In the local Swarm node, data is split up into pieces called chunks with limited sizes (max 4K), which are the basic units of storage and retrieval system [114]. After chunks are stored in a Merkle tree in the local node and addressed with the hash of their contents, the chunks are distributed across the network. The hash references of the data chunks that makes a content are packaged in to a single chunk with the Merkle root hash of all the chunks as depicted in in Figure 3.7. The final root hash acts as a 32-byte reference address, and by using this address the file is retrievable while ensuring data integrity of the file via Merkle tree. Since the network layer and nodes know only the chunks, they do not have information in the file level, which increases the security of Swarm.

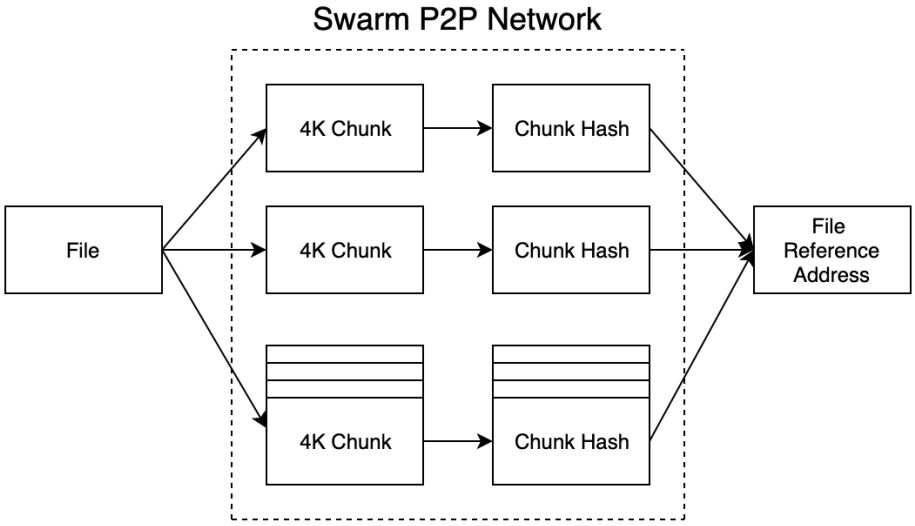

Figure 3.7: Swarm File Storage Process

Once a content is uploaded to the local node, that node syncs the chunks of data with other nodes on the network. Consequently, that content will be available on the network even the original node goes offline. However it should be noted that, persistence of the content is not guaranteed until the incentive mechanism of Swarm is implemented, so it should not be considered as a safe storage till then [114].

3.5.4. Ethereum Name Service

In Swarm, a content is retrievable with its unique addresses derived from the cryptographic hash of the data. This address derived from hash is not user friendly, on top of it, when the content is changed, the hash of the content and its address changes accordingly. To solve this issue, Ethereum Web 3.0 stack has another native base layer service called Ethereum Name Service (ENS), which acts as a name service. ENS is basically the Ethereum equivalent of the Domain Name Service (DNS) in the WWW. ENS provides a secure and decentralized way to address resources both on and off the Ethereum blockchain, such as Ethereum addresses and content hashes, using human-readable names with ".eth" extension like "alice.eth" [116]. ENS is based on a suite of smart contracts in Ethereum mainnet and in order to use ENS to resolve names to Swarm addresses, that swarm node has to connect to the Mainnet over an Ethereum client [114].

3.5.5. Message Communications (Whisper)

Communication protocols are necessary for enabling applications and users to send messages in the network. In the current web architecture, this is achieved by central servers and protocols. In order to decentralize message communications, Ethereum Web 3.0 stack has a native layer service, called Whisper. Whisper is a peer-to-peer communication protocol to enable DApps and its users a communication channel in Ethereum network. It is an identity-based communication system, and provides private and secure communication directly between nodes, using PKC [117].

3.5.6. Data Feeds (Oracle)

Blockchains, and inherently Ethereum, are deterministically verifiable architectures, which means, each node can verify the entire blockchain at any time. This deterministic model is ensured by allowing only on-chain data during transaction executions. In Ethereum, a smart contract can use only the data which is passed with a transaction. Since this unreliable external source may give different answers to same queries in different times, allowing off-chain data breaks this deterministic model. This is also the reason why Ethereum smart contract languages and EVM architecture does not support built-in randomness. Since most of the web applications use real word data, this limited data feed restricts the usefulness of the smart contracts and DApps.

In order to make useful DApps, Ethereum requires a data feed system to be able to get data from off-chain systems. This data feed system is called as Oracle, which is a service that provides data external to Ethereum, and feed that data into smart contracts or DApps. Oracles might be used to provide real world information such as random numbers, exchange rate data, sport events, weather data, statistics, etc. [84].

To be used as a reliable source of information, oracles must provide certain guarantees for the data they provide. Ideally, like other decentralized components of Ethereum Web 3.0 architecture, oracles should be trustless, too. Nevertheless, there are centralized oracles as well. Oraclize is a centralized oracle which uses authenticity proofs in order to demonstrate the data fetched from the original data-source is genuine and untampered, thus acting as some sort of an trustless intermediary [118]. ChainLink

offers a reliable decentralized oracle service, which ensures the integrity of the oracles with validation, reputation, certification, and contract-upgrade services [119].

CHAPTER 4

SYSTEM DESIGN AND ARCHITECTURE

Using blockchain (permissionless or permissioned) is suitable when it is desired that mutually mistrusting parties be able to interact and change the state of the system, without the need for a TTP [5]. As we know already, there are three types of blockchain networks regarding how they accept nodes for the network consensus: permissionless blockchain, permissioned blockchain, and consortium blockchain. Each type of blockchain network has its own advantages and disadvantages as listed in Table 3.1. This study aims to provide a secure framework for content sharing, open to anybody, without a limitation or censorship. Although efficiency in permissionless blockchain is low, and there are transaction costs; nonetheless, it ensures immutability, reliability, transparency, and decentralization. Therefore, permissionless blockchain was the reasonable approach for our proof of concept.

In order to maintain decentralization of our application logic, we utilized smart contracts to create proof of concept decentralized application (DApp) of this study. There are several permissionless smart contract and decentralized application platforms, such as Ethereum, Cardano, EOS etc. Nevertheless, we opted for Ethereum as the development platform for our proof of concept because of several reasons: First, Ethereum is the most commonly used [120] and most popular [121] technology among its competitors. Second, Ethereum beats its competitors in terms of usability, documentation, and development support [122]. Third, there are more DApps created with Ethereum compared to other platforms [108]. Fourth, Ethereum is not a mere decentralized application platform, but it provides a whole ecosystem for decentralized web. Fifth, as the first of its kind, Ethereum platform is relatively mature, and has a large and active community. Finally, there are several real-world applications of Ethereum in use. Although Ethereum blockchain can be setup as a private blockchain inside an access-controlled network by creating a new genesis block and filtering the miners, we did stick by permissionless public blockchain.

After we observed the suitability of the blockchains and smart contracts for content sharing, in order to investigate the subject further, we have developed a proof of concept DApp which enables a channel for securely sharing digital content between parties that do not necessarily trust each other, as defined a goal of this study. In this chapter we thoroughly describe and explain the design of this proof of concept.

As illustrated in Figure 3.5, a typical Ethereum DApp architecture has several components such as the DApp browser, blockchain, smart contracts, and potentially a

few of the services of the Ethereum Web 3.0 stack. Before we started design phase, we conducted a simple requirement analysis. Namely, we elicited functional and non-functional requirements, defined user roles and use cases, chose the technologies to use for our software with the justifications of the rationale behind them. Subsequently, we designed smart contracts, and finally came up with the overall architectural design of the system.

4.1. Software Requirements

For the use of our DApp, we assumed users had carried out following procedures in advance, thus we excluded these actions from our design:

- Install a modern web browser such as Chrome, Firefox, or Opera.
- Install the MetaMask extension and create a passphrase to use the vault.
- Connect to Ethereum network (personal test network for this proof of concept).
- Create or import an Ethereum account in MetaMask.
- Transfer reasonable amount of Ether to the Ethereum account to be able to make transactions (for this proof of concept users have already pre-installed Ether on their accounts).
- Run a Swarm node.

Using MetaMask, a user can unlock her Ethereum account and interact with Ethereum network and DApps. Therefore, there is no dedicated user registration or user login requirements specific to our DApp. Likewise, authentication procedure is managed by MetaMask.

4.1.1. User Roles

In our content sharing DApp, there are two types of user roles:

- **Seller (Sender).** Seller is the user who publishes the digital content on sale. Note that the content can be free. So according to price and context, we can use the terms ‘seller’, ‘sender’, and ‘publisher’ interchangeably.
- **Buyer (Recipient).** Buyer is the user who fulfill requirements to access the digital content on sale. Similarly, according to price and context, we can use ‘buyer’, ‘recipient’, and ‘consumer’ interchangeably.

4.1.2. Functional Requirments

In order to let users securely exchange content, our proof of concept DApp must satisfy the following functional requirements (FR):

FR1. The DApp shall be accessible and usable by any Ethereum account.

FR2. The DApp shall let users view content on sale.

FR3. The DApp shall let a seller to put digital content up for sale.

FR4. The DApp shall let a buyer to buy digital content on sale.

FR5. The DApp shall let a seller to abort the sale before it is bought.

FR6. The DApp shall let a buyer to pay for an open sale.

FR7. The DApp shall let a seller to encrypt digital content.

FR8. The DApp shall let a seller to upload digital content.

FR9. The DApp shall let a buyer to decrypt digital content.

FR10. The DApp shall let a buyer to download digital content.

FR11. The DApp shall let a buyer to confirm receipt.

FR12. The DApp shall let a seller get paid.

FR13. The DApp shall act as an escrow to protect both buyers and sellers against fraudulent users.

4.1.3. Non-functional Requirements

We elicited functional and non-functional requirements (NFR) together, so they actually reinforce each other. Accordingly, we came up with several non-functional requirements according to different quality attributes.

NFR1. (**Confidentiality**). The DApp shall enforce confidentiality of the content by providing access to unencrypted content to only authorized users using PKC.

NFR2. (**Integrity**). The system shall preserve integrity of the data in the front-end, blockchain, and decentralized storage.

NFR3. (**Availability**). The system shall be available for all Ethereum users and the content shall be available only for authorized users.

NFR4. (**Privacy**). The system shall maintain sufficient privacy for the users.

NFR5. (**Usability**). The system shall be sufficiently easy to use with a simple and descriptive front-end.

NFR6. (**Transparency and Auditability**). The system shall let the transaction data be transparent and auditable.

NFR7. (**Costs**). The DApp usage shall be free of charge, and transaction costs shall be sufficiently low. The data written to blockchain shall be kept minimum, by storing large data in the decentralized storage.

NFR8. (**Reliability**). The system shall ensure confidence in the correctness of executed transactions.

NFR9. (**Scalability**). The system shall allow several sale and buy actions to be made at the same time with a sufficient performance.

NFR10. (**Performance**). The system shall provide a fast frontend interaction, and the user actions shall be completed sufficiently fast.

4.1.4. Use Cases

Users achieve their goals using certain actions they are able to perform as listed in functional requirements. Different ways a user can interact with the system is depicted in a use case diagram in Figure 4.1.

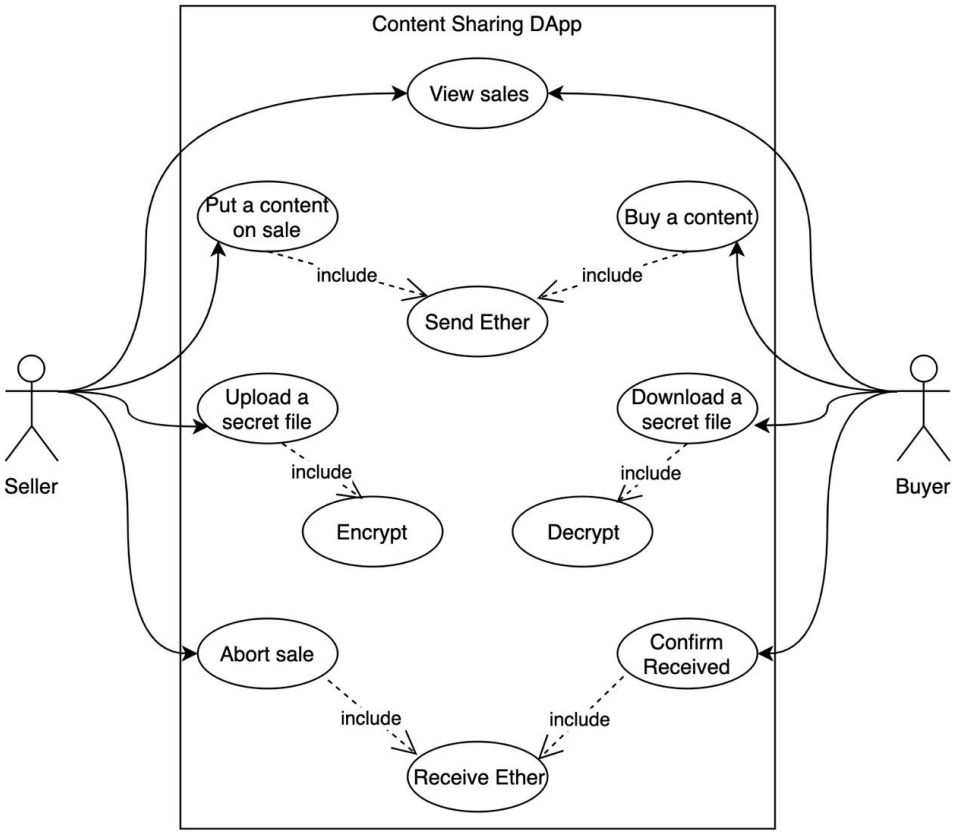

Figure 4.1: Use Case diagram of the Content Sharing DApp

4.2. Data Management

A buyer needs to securely deliver the digital content in order to finalize the trade. Dealing with the data in our DApp has two aspects, providing confidentiality and storage. First, the digital content must be encrypted so as to be only decrypted by a buyer. This is the one of the most crucial factors when sharing a secret in Ethereum. Second, digital content itself, and large sized data related with the exchange must be stored off-chain, in a decentralized storage, i.e., Swarm. Likewise, this is one of the most crucial factors for the overall usability of the blockchain.

4.2.1. Data Confidentiality

Confidentiality of the digital content is the most critical non-functional requirement for our content share framework. As discussed before, Ethereum does not have a built-in privacy support for the data stored in its blockchain. Smart contract data and logic is also transparent. Consequently, to enable the information to be accessed only by authorized users, we needed to enforce confidentiality off-chain. For Alice and Bob to exchange secret information with each other, without revealing the data to any third party, we utilized cryptography. For this purpose, we could either use symmetric key cryptography or PKC.

If symmetric key cryptography was used, sensitive data would be encrypted and decrypted with the same secret key, thus both Alice and Bob had to possess the key before the transaction. This approach requires off-channel information exchange in order to come to an agreement for the secret key. When Alice and Bob do not know each other, it is not very feasible to use a pre-shared key for the encryption. Instead, we aimed to provide a framework where Alice and Bob would not need to know or trust each other. Accordingly, we used PKC in our design. In PKC, a key pair is generated and when a secret is encrypted with the public key, it can only be decrypted with the private key. This mechanism also suits our content sharing framework. Bob writes his public key to the smart contract; Alice use this key to encrypt the secret and write the encrypted secret back to the smart contract. Bob decrypts the encrypted secret with his private key and access the secret, thus the content which is accessible only with this secret over local Swarm node.

Since Alice and Bob use an Ethereum account to interact with our DApp, they already have ECDSA key pairs for their Ethereum accounts. Using these keys for encryption/decryption of the secret is the first thing that came to our mind. Yet, there are three issues with this approach. First, it is not a good practice to use the same private key for both encryption and transaction signing. This is why, for example, GnuPG [123], which is an implementation of the OpenPGP standard, creates a master key for signing and a different sub-key for encryption. Second, dealing with the key pairs that belongs to an account that holds actual Ether is risky. Normally, MetaMask acts as a secure identity vault and deals with the private keys of Ethereum accounts while signing the transactions for Ethereum blockchain and DApps. MetaMask does not offer a mean to use private keys for decryption. So, we need to deal with the Ethereum account's private keys from the front-end, which can raise security concerns. Third, using the same key pair for every buy transaction would yield several pieces of

encrypted content to be decrypted by a malicious user once the private key is somehow disclosed in the future. To ensure forward secrecy, the user needs to create a new Ethereum account and transfer Ether to this account each time she buys a content, which is not very practical. Because of the security and usability concerns, we decided not to use Ethereum account key pairs for ensuring the confidentiality of the digital content.

Rather than using existing Ethereum key pairs, we thought we could create a new PGP key pair for each buy action. By this means, we ensured that the same key would be newer used twice, and hence provided forward secrecy without the need for creating new Ethereum accounts for each buy action. In this context, we can think of our key pair generation strategy as an occasion for achieving forward secrecy, similar to ephemeral key exchange mechanisms, such as Ephemeral Diffie-Hellman (DHE) and Ephemeral Elliptic-curve Diffie-Hellman (ECDHE).

PKC in our DApp was achieved by using a JavaScript library to generate the key pairs and perform encryption and decryption of the secret from the front-end via OpenPGP protocol. During our research, we found out a proven open source library called OpenPGP.js[1] , which is a JavaScript implementation of the OpenPGP protocol. Although OpenPGP[2] is a protocol mainly used for end-to-end e-mail encryption, it can also be used as a mean for encrypted messaging. Having said that, OpenPGP.js library is used and maintained by ProtonMail[3], an e-mail service provider specialized in end-to-end encryption of the e-mails. OpenPGP.js library has also passed an independent security audit conducted by Cure53 [124]. Namely, we confirmed this library is safe to use in our DApp. In order to generate an OpenPGP key pair, OpenPGP.js requires the user to fill three fields: name, e-mail, and passphrase. The PGP private key is encrypted with the passphrase so actually it behaves like a keystore file. Thus, saving the private key file in the owner's local directory is relatively safe, as the file cannot be decrypted without the passphrase.

4.2.2. Data Storage

Smart contracts in Ethereum can store data in order to perform the application logic of the DApp. Our smart contracts need several state variables that are stored in the blockchain, such as price of the content, seller, buyer, name and description of the content, a link to content, and a link to public key of the buyer. Storing data in blockchain requires gas, so we stored as much data as we can in off-chain, specifically the files. Without compromising the decentralized approach of the DApp, we could store the large data in a decentralized storage system, such as Swarm and IPFS. Note that these large data and files are not actually required for blockchain consensus. Thus,

---

[1] https://openpgpjs.org

[2] https://openpgp.org

[3] https://protonmail.com

rather than the actual content itself, we stored the reference address of the content on the blockchain.

Either of IPFS and Swarm could have been used in this study. As a matter of fact, compared to Swarm, IPFS is more mature, it has a larger community, and has been adopted by more projects and studies. According to our literature review, there are very limited number of studies that choose Swarm as the decentralized storage system. Nevertheless, we opted for Swarm, due to several key advantages of it over IPFS: First, according to Swarm documentation[114], Swarm is a native base layer service of Ethereum, this is why a Swarm account is actually an Ethereum account and Swarm is intrinsically linked to the Ethereum blockchain as a storage network. Second, Swarm provides a built-in incentive layer in its architecture, which makes it possible to upload a content and go offline, though this incentive mechanism is not fully implemented yet. Third, Swarm reference addresses are 32 bytes, which are compatible with 256-bit word size of the EVM[77]. Thus, Swarm references can be stored in Byte32 value type, which is cheaper than storing in a string. IPFS reference address are larger than 32 bytes. Finally, and most importantly, Swarm has built-in encryption and access control mechanisms, which are useful when dealing with sensitive data. In conclusion, assured the security and immutability of our off-chain data utilizing Swarm.

Table 4.1: State variables of the smart contract

```
uint public price;
address public seller;     //Ethereum address
address public buyer;      //Ethereum address
string public name;
string public description;
bytes32 public secretHashAddress;     //Swarm reference address
bytes32 public contentHashAddress;    //Swarm reference address
bytes32 public buyerPublicKey;        //Swarm reference address
```

For interacting with the Swarm network in order to upload and download data, we used Erebos[4] API. Erebos is a JavaScript client that is supported by Swarm. When a file or directory is uploaded to Swarm, it gives a reference address to the uploaded content. Swarm references are 32 bytes. So, technically, if a data is larger than 32 bytes, it is cheaper to store it in Swarm, compared to storing it in blockchain. The state variables that store data of a content share action in our smart contract are listed in Table 4.1 with their Solidity value types and visibilities. Note that accounts in Ethereum network are identified by Ethereum addresses, which are 20 bytes long. We also did not store the name and description of the content in Swarm, since we assume these strings would be probably less than 32 bytes. Nevertheless, strings are arbitrary length values, so the length of the strings would be limited in the front-end, in order

[4] https://erebos.js.org

not to let user to spend unnecessary amount of gas for the transaction. Typically, it is a good idea to store the content, secret file, and OpenPGP standard public key to Swarm. So, in the smart contract, only reference addresses of these files are stored.

4.3. Smart Contracts

Smart contracts enforce the application logic in DApps. We have two smart contracts for our DApp. First smart contract handles the sell/buy processes among users. The second smart contract handles the information and deployment of the first smart contract in to separate instances.

Solidity is the de facto programming language for Ethereum smart contracts. Although Vyper is the emerging security oriented smart contract language, and it provides a simple programming approach to be more resistant to human errors, it is still an experimental programming language with only beta versions released yet [95]. On the other hand, Solidity is more mature, has a large community, has many independent documentations with lots of examples. Accordingly, we programmed out smart contracts using Solidity. As discussed in Section 3.3.3, smart contracts have several potential vulnerabilities and smart contract security is a critical aspect to be careful about. There are tools to conduct security tests for Solidity source codes. So, in order to be in the safe side, we had our smart contracts tested for security vulnerabilities.

4.3.1. Reasonably Fair Exchange via Double Escrow Smart Contract

In conventional web based digital exchange platforms, interest of buyers and sellers are protected by the website owner, or privileged users with delegated powers, which act in disputed transactions. Most of the time, publishers upload their digital content to the web app's server, leaving the digital content in the hands of the web app owner. When a user purchases the content, buyer downloads the files from the website. This process protects the buyer from fraud. In this scenario, payments are done over banking systems, which protects the seller from fraud. Most of the time, web apps keep the money for certain amount of time before they deliver it to the seller, in case there is a dispute. These are all examples of current applications of fair exchange using TTPs.

In private blockchain based sharing platforms, only permissioned users can join and interact with the system. This yields a framework where users trust each other, because they share the same business interest, or there is a centralized authority that maintains the order against malicious behavior. In this scenario, a remote purchase is intrinsically safe.

In decentralized applications on permissionless blockchains, where the blockchain data is transparent, access control can be provided via PKC. However, ensuring safe remote purchase, i.e., *fair exchange* is still not trivial, in a network where users do not trust each other. When Alice puts her digital content on sale, she wants it to be accessible by only buyers. Thus, the digital content on the blockchain must be

encrypted until the buyer downloads and decrypts it. After Bob makes the payment, there is a possibility that he finds out Alice acted maliciously and sent garbage content instead. On the other hand, if the digital content is delivered to the Bob before he makes the payment, then a malicious Alice can decide not to pay at all. In this scenario there is no protection mechanism for either of the buyer or seller against fraud. This problem is called fair exchange problem in electronic commerce. For an exchange to be fair, either both parties should get what they want, or neither party should get anything.

Having a reputation system for the users is a common practice in content sharing platforms. For example, in eBay, people mostly prefer buying goods from the reputable sellers with high points. This is an incentivized approach, where sellers sell more goods when they have high reputation, thus do not act maliciously because users can give negative feedback and lower their points. Reputation based fairness solutions are proposed in the context of fair exchange [125]. Although a reputation system can help achieving a somewhat fair exchange, it is stated that reputations systems have limited applicability due to uncertainty of the definition of reputation for the new users [126]. Thus, it could act only as an auxiliary function for our DApp, which can be applied in the future.

In Ethereum, transactions are first broadcasted to the network and a malicious user can get information before that transaction is verified and mined. So, implementing an escrow mechanism with deposits makes sense in the context of blockchain based exchanges. Using escrow services for taking deposits in blockchain is a method studied in several work [54] [50] [53]. Enforcing monetary penalties to malicious users or mediators [47] [50] is proposed as an approach for fair exchange protocols. Likewise, there are protocols that enforce parties to deposit predefined amount of money as an incentive to achieve fair exchange [52] [53] [54] [55]. Similarly, we implemented our two-party fair exchange protocol using escrows and deposits, though using smart contracts.

Asokan et al. [41] defined three type of electronic goods as the exchangeable items in their generic protocol for fair exchange: confidential data, public data, and payments. Similarly, our approach to fair exchange of digital content has all three type of exchangeable items: encrypted content and its symmetric key are the confidential data, smart contract data is the public data, and cryptocurrency transfer is the payment. Payment and confidential data should be tied to each other in order to achieve fairness. On the other hand, revealing of the public data does not affect the fairness.

Escrow services are the agents where at least one of the parties deposit a certain amount of money before their transaction. According to Hu et. al, adopting online escrow services as a remedy for internet fraud in customer to customer online trading systems is a viable approach [127]. They determined the demand for escrow services and conducted a numerical study for optimal prices for the service. According to their study, online escrow agents effectually prevent fraud.

Our escrow logic is handled and autonomously enforced by a smart contract. Since smart contracts can transfer and receive Ether, they are inherently suitable to act as an escrow stakeholder between seller and buyer. In addition, smart contract logic and data

is transparent and can be audited by anybody who is interested. Hence, our smart contract is both the trustless mediator and escrow service. This approach is different from other Bitcoin based escrow propositions such as the work of Goldfeder et al. [50], where the mediator is offline, and the protocol is optimistic. Our escrow protocol, is active on deposits, active on withdrawals, is not optimistic, is secure, is not externally-hiding, and is not internally hiding due to the fact that Ethereum blockchain does not provide privacy, according to the definitions given in [50].

When we decided to implement an escrow feature via a smart contract, we needed to decide the specifications of our escrow logic, such as the amount of deposit, which parties should make deposit, when to make deposit, and when to return deposit. In order to ensure the fairness of the exchange, as the problem described above, we need both seller and buyer to make certain amount of deposit, as an incentive to act honestly. After our research, we found out that there are already proposed schemes for double escrow services in blockchain systems. For example, Zimbeck proposed a double deposit escrow in Bitcoin network [128]. Also, Asgaonkar and Krishnamachari proposed a dual-deposit escrow protocol for a cheat-proof delivery [129].

For our smart contact, we adapted a double escrow mechanism based on the "Safe Remote Purchase" contract example in Solidity Documentation [130]. Escrow smart contract is developed as stateful, with the following states: Created, Locked, and Inactive. The amount of deposit that the seller needs to make is double the amount of the price of the digital content that the seller put on sale. Likewise, the total amount of deposit that the buyer needs to make is again double the amount of the price of the digital content she wants to buy. When the transaction ends successfully, seller receives full deposit plus the content price, and the buyer receives deposit minus content price. Safety and liveness of the purchase process is directly proportional to the amount of the deposit [129]. Nonetheless, we assumed double the value of the price is an optimal deposit amount for us.

The rationale behind choosing this deposit amount is to equally incentivize both the buyer and seller to finish the purchase process, otherwise they lose same amount of money. Assume Alice put a digital content on sale with the price of 10 Ether, and Bob wants to buy the content. So they both deposit 20 Ether to the smart contract. Assume Alice deposits 10 Ether, then she may not send the content, and Bob would lose more money than Alice. Assume Bob's deposit is 10 Ether, then he may not confirm that he received the content, then Alice lose 20 Ether, and Bob does not lose anything because he deposited the same amount of money with the price of the content. As can be seen, by choosing this deposit amount, we make the seller and buyer to take the same amount of risk, thus equally committing to the purchase process.

Since we have a commitment scheme, which means both parties deposit certain amount of money to the mediator escrow smart contract before the exchange, we achieved a two-party fair exchange protocol with incentive, of which the penalty amount is predetermined, and equal for both of the parties.

Our equal incentive approach actually assures exchange fairness for rational users, as it is described in studies on incentive game theory based secret sharing schemes [131] [132]. These works do not define users as pure malicious or pure honest, but with the

assumption that they are rational. A rational party acts honestly when he cannot gain advantage over other party, however acts maliciously if he can gain advantage over the other party. This means, rational parties will act honestly if there is no incentive or negative incentive for fraud. In other words, when a rational buyer deposits more money than the price of the digital content, or a rational seller deposits more money than he would receive as a payment, they will act honestly. They will be equally committed to finish the transaction and they would not deviate from the protocol, because it is against their self-interest. A purely malicious user would be discouraged from using the system, and a rational malicious user would act honestly.

Since Alice and Bob are not supposed to make any off-chain communication, and all the communication is done via the smart contract, our fair exchange protocol is not optimistic. Nevertheless, it does not cause any extra communicational overhead, as it is in conventional TTPs. Instead of any transaction addressed to either Alice or Bob, they both send transactions to the escrow. Since reading blockchain data is free, there is no need for the smart contract to send any message to any parties, so that actually the number of transactions does not change whether the mediator exist or not. However, the overhead this mediator brings is actually gas (i.e., Ether) and time. Optimistic fair exchange protocols require TTPs for aborting the transaction or triggering a dispute. Our smart contract enables abort action, and thanks to the penalty scheme, we would not need dispute mechanism, with the assumption of users being rational.

Our escrow smart contract is the trustless mediator, acting on predefined rules so that neither Alice nor Bob can withdraw money unilaterally. There is no chance that the mediator act maliciously because there is no one controls it, namely, the mediator cannot have access the digital content, cannot send deposits to Chuck, cannot seize the deposits unless one of the parties act maliciously, and cannot favor one of the parties against other.

We should also note that, although this double escrow scheme discourages users act maliciously as they would lose their deposit, an irrational user may want to make other party lose money at the cost of him losing his money as well. The architecture can be modified so that a seller can abort the sale any time before the delivery is confirmed by the seller which would send the deposits back to both of the parties. Or a time limit can be integrated in to the system so that both sides need to take action within the time limit. Although these modifications can protect users in some ways, they have their own drawbacks as well. When a seller aborts the sale after the buyer send money, the buyer would lose time and her money that would be usable for that time being. When there is time locks, if a user has a DoS attack or is not available for some reason, she might lose money even if she did not act maliciously. In conclusion, there can be other approaches to the problem, nevertheless, the protocol we implemented can be considered as secure and fair with the assumption that the users act rationally.

The two-party fair exchange protocol of safe remote purchase via double escrow mechanism in our DApp is described in the following steps:

- **Put Content on Sale**.

  Seller creates a purchase contract with the price, say 10 Ether. Seller deposits 20 Ether. Contract state is “Created” and anybody can buy the content at this moment. Contract balance is 20 Ether.

- **Abort Sale.**

  If the contract state is “Created”, i.e., nobody bought the content yet, seller can abort the sale. This is an optional functionality for enabling seller to retrieve her deposit back if nobody buys the content or the seller does not want to sell the content anymore. When the sale is aborted, contract state becomes “Inactive”, so it cannot be bought any more. Escrow contract transfers 20 Ether back to seller. Contract balance is zero Ether in the end.

- **Buy Content.**

  Buyer sends 20 Ether to the contract. Buyer stores her public key to smart contract. Contract state is now “Locked”, which means only this buyer can continue with the buying process from now on. Escrow contract balance is 40 Ether now.

- **Deliver Content.**

  Seller sees the contract is bought. Seller encrypts the secret with buyer’s public key and stores the encrypted secret in to the blockchain, meaning she delivers the encrypted content to the buyer. Contract state is still “Locked”, and Contract balance is still 40 Ether.

- **Confirm Received**.

  Buyer sees the content is delivered by the seller. Buyer download and decrypts the content with her own private key. Buyer confirms the content received. Contract state becomes “Inactive”. Contract transfers 30 Ether to seller, and 10 Ether to buyer. Eventually contract balance becomes zero, and the extra amount of Ether the seller receives is actually the price of the digital content she is selling in the first place.

Sell process and buy process with our escrow scheme via a stateful smart contract implementation are depicted in Figure 4.2 and Figure 4.3, respectively.

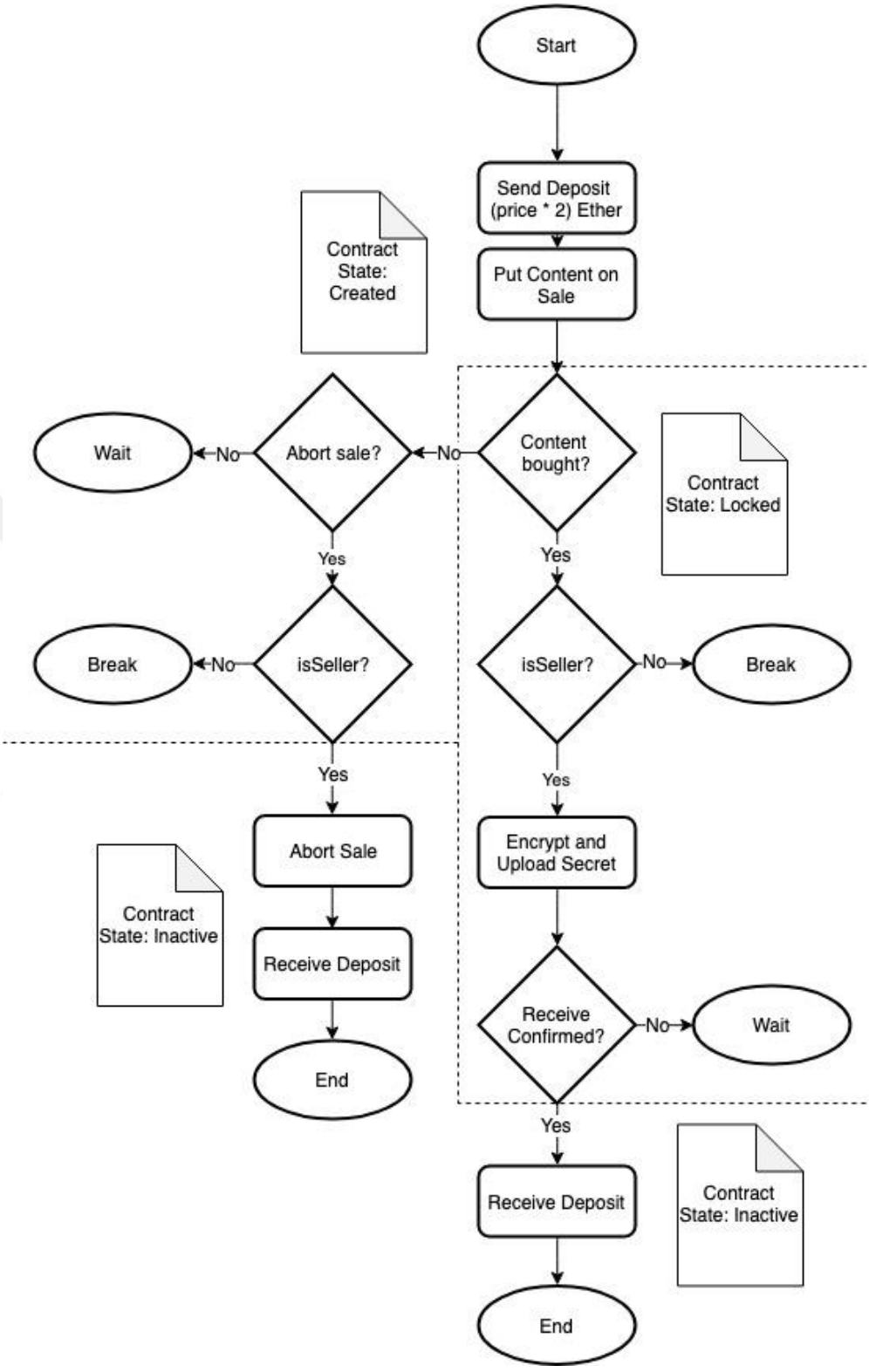

Figure 4.2: Seller Flowchart

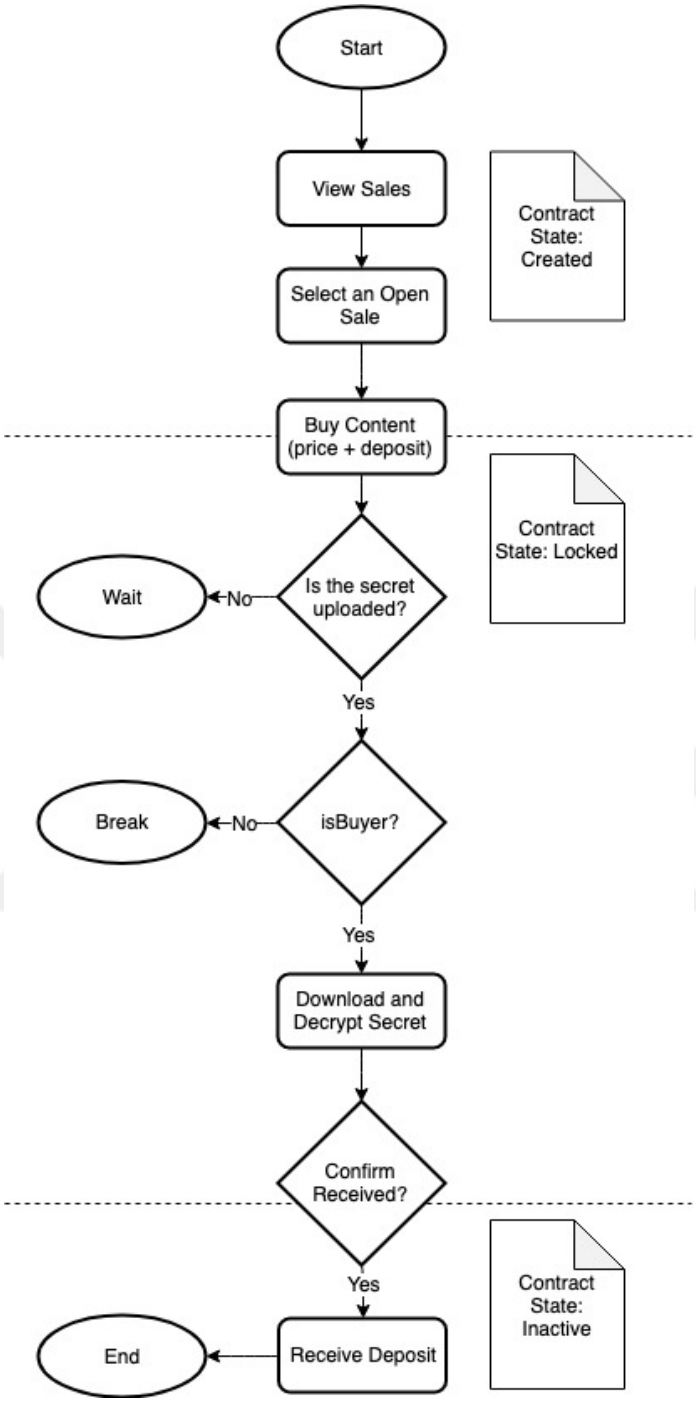

Figure 4.3: Buyer Flowchart

4.3.2. Management Contract

The escrow smart contract described above, handles the sale and purchase logic between a buyer and seller. In order to allow multiple simultaneous sale/buy actions, we need to have multiple contents with its specific state variables, such as buyer address, seller address, price, etc. in our smart contract. It is possible to manage separate content sales by storing each sale in an array. However, there is a problem

with this approach. Since our smart contract acts as an escrow, it stores deposits from both the buyer and seller. So, if we store every escrow in one array, our single smart contract would hold total amount of deposits coming from each sale. Keeping all the escrows in one place might be risky. On the other hand, if we deploy a new escrow contract for each sale, seller needs to pay more Ether for the sale initial transaction, because contract creation would costs an extra 32000 gas [77]. According to ETH Gas Station[5], the current (as of August 2019) standard gas price is 3 Gwei. So, deploying a new contract is going to additionally cost no more than 0.02 USD, which is actually negligible if you sell a content for, let's say 100 USD (see Section 6.3 for detailed cost analysis). In any case, separating escrow logic from platform logic would decrease the complexity of the smart contract, and we want to keep the smart contracts as simple as possible as it is recommended by the best practice. Given the above, a seller creates a new escrow contract for each sale using the management contract, so it acts like a contract factory. For the purpose of manageability, management contract stores an array of addresses of all deployed escrow contracts as illustrated in Figure 4.4. Deployed sale contracts can be listed in the frontend by reading addresses of the contracts from the *deployedContracts* array and getting the details of each escrow purchase contract.

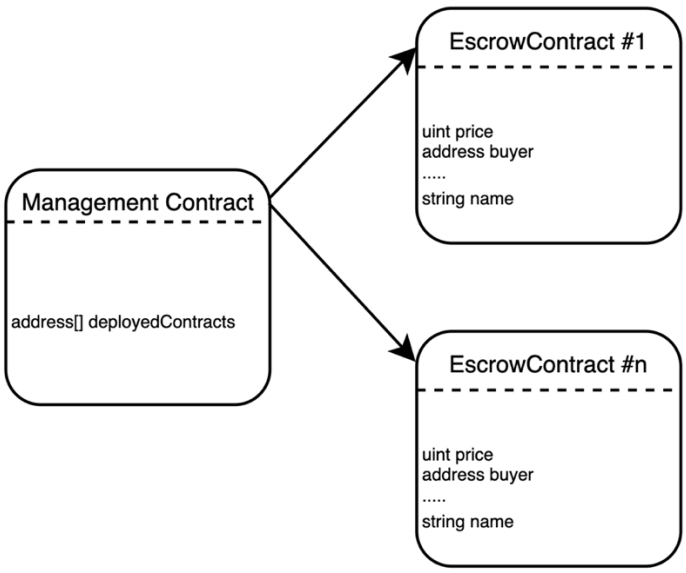

Figure 4.4: Management Contract

4.4. Final System Architecture

In this section we describe the final system architecture of our secure content sharing DApp. Figure 4.5 depicts the overall architecture, and it shows the components of the

[5] https://ethgasstation.info/

system, as well as how they interact each other in detail. As can be seen, there are two main layers in the architecture: presentation layer (front-end), and database and application layers (back-end). A user interacts with all of these components either directly or indirectly in order to sell and buy content in our decentralized secure content sharing framework.

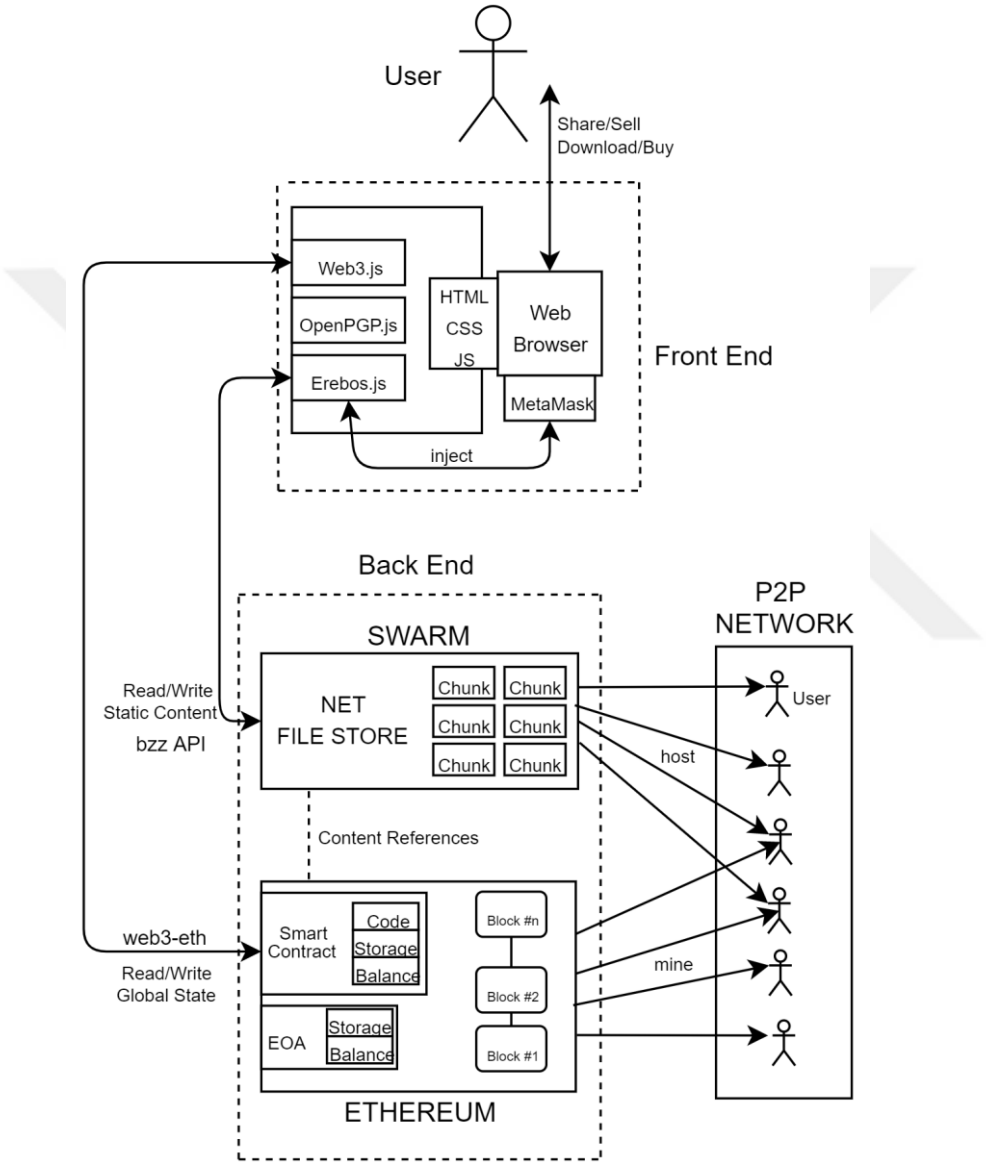

Figure 4.5: Final System Architecture

Front-end software is a collection of components that makes a graphical user interface which enables an end user to use the DApp through their web browser. With CSS and HTML files, this interface lets the end user send information to frontend, and displays the contents and information back to the user. It also has JavaScript libraries acting as client-side tools and communication interfaces to interact with the back-end. Erebos.js

library is used to read/write static content from/to local Swarm node though bzz API. This is used to store the large files in smart contracts as reference addresses. OpenPGP.js library provides the necessary PKC primitives, such as key generation, encryption, and decryption. This is used to let a seller send a secret to the buyer without any pre-shared information. Web3.js library enables a link between front-end and a local or remote Ethereum node. As a matter of fact, Ethereum blockchain provides a JSON RPC API to interact with a local or remote node through RPC calls, and Web3.js is a JavaScript wrapper that enables a developer friendly access to several functionalities, without the need to write raw JSON RPC calls. All the read/write global state from/to Ethereum node actions, interactions with smart contracts, signing and sending transactions, etc. are handled by Web3.js library. Web3.js utilizes the contract ABIs, which are JavaScript interpretation layers to interact with the deployed contracts in the blockchain from the front-end.

MetaMask runs integrated with our front-end software as a component for blockchain interaction. It is the de facto standard for DApp browsing, used for account and key management, user authorization, transaction signing, and interaction of end users with a local or remote Ethereum node using Web3.js. Since it is a browser extension, it can interact with every webpage that user browses. It injects a Web3 object, which is available in JavaScript context, to loading pages, thus enabling them to be used by the JavaScript code of the DApp. Detection of the current user and other account actions are handled by this web3 object.

Back-end software represents the database and application layers. Database and data access layers consist of the immutable distributed ledger of Ethereum blockchain and the Swarm storage system. All the transactions and data used in application logic is stored in blockchain, either directly or with reference addresses. Swarm stores the large files in chunks, which are accessible by the reference addresses stored in the blockchain. Basically, blockchain acts as the immutable database, and Swarm acts as the content delivery network for our DApp. The application layer is actually the smart contracts that manage the business logic of the application. In addition to smart contracts and transactions ledger, blockchain also stores EOA accounts for transactions, triggered by the DApp and signed by MetaMask. All the back-end software, i.e., Ethereum blockchain and Swarm, operate on a peer-to-peer network. Ethereum nodes mine Ethereum blockchain for validation and consensus purposes. Swarm nodes host referenced chunks of the files. Note that, in order to interact with peer-to-peer networks, a user need to be a peer, as well. For Ethereum, MetaMask takes this burden from the user by directing the transactions to secure Ethereum nodes provided by Infura[6], so end-user does not need to be an Ethereum node to interact with the network. For Swarm, although there is a possibility to use a public gateway[7] to upload/retrieve data to/from Swarm, it is not safe to use this gateway for sensitive data. Besides, accessing access-controlled content is enabled only when using a local Swarm node [114]. Hence, the users of the proof of concept are required to run a local

[6] https://infura.io

[7] https://swarm-gateways.net

Swarm node as a peer in the peer-to-peer network, which is expressed in the final system architecture (Figure 4.5).

By utilizing blockchain, and storing files in Swarm, rather than cloud or local storages, we reached a high level of decentralization for our DApp, as a matter of fact, only central point of failure for our DApp is that it is served from a web server. In order to achieve a full decentralization, we can host the front-end software, i.e., HTML, CSS, and JavaScript files in the Swarm. Swarm supports only static pages, accordingly the components in the front-end need to be made static. This can be achieved via a JavaScript bundler, such as Webpack. Once all the files deployed to a static library, proposed DApp is able to be served by Swarm. When a user wants to view previous sales, the existing sales can be reached by the front-end software via the array of escrow-purchase contracts stored in the management contract. Information regarding these deployed contracts needs to be loaded each time a user accesses the DApp. Nevertheless, this content exchange platform is not going to be able to utilize dynamic web pages, so it would have limited functionality relative to more complex designs. Essentially, instead of relying a server, and by hosting our DApp in Swarm, we would decentralize our DApp even more, with limitation of a relatively simple architecture. Consequently, when the proposed DApp is hosted in Swarm, it becomes resistant to DoS attacks and censorship, and maintaining security and zero downtime, thus assuring the availability of the DApp to anybody without any permission or limit.

Note that when a seller sets the price to zero, our prosed DApp can be used to securely share information and digital content between any two parties, with end-to-end encryption. Since Ethereum blockchain data is transparent, and all the other operations are done in client side via JavaScript scripts, both sides can verify the security of the application.

CHAPTER 5

IMPLEMENTATION AND VALIDATION

After we conducted requirement analysis and system design of the proof of concept, we implemented the proposed DApp. This chapter gives information about this implementation process. First the development environment is described, then implementation of the smart contracts and front-end software are explained. Finally, in the last section, the validation process of the implemented DApp according to the functional requirements and design specifications is described. Note that since this is a proof of concept, we connected our DApp to a private test network, instead of the Ethereum mainnet.

5.1. Development Environment

The tools we used for the development of the DApp are as follows: Geth, Ganache, Truffle, MetaMask, and WebStorm.

Geth[8] **(**Go Ethereum**)** is the official implementation of the Ethereum protocol, written in Go language. It provides a command line interface to run a node connected to Ethereum blockchain, through which, the mining process, and other interactions with the blockchain, such as account creation, transaction sending, etc. are enabled. It also allows to connect any public or private Ethereum blockchain. Ethereum and Swarm accounts are created using Geth.

Ganache[9] is an in-memory personal Ethereum blockchain for developers to run tests on. It provides a graphical user interface to enable easier access to global state of the private blockchain. Accounts, blocks, transactions, contracts, events, and logs can be inspected from this user interface. Through Ganache, all the software tests can be done before actually deploying the DApp to the Ethereum mainnet or other test networks. Ganache was used as the Ethereum blockchain throughout implementation and testing. As can be seen in Figure 5.1, Ganache provides pre-created accounts with balances to

[8] https://geth.ethereum.org

[9] https://trufflesuite.com/ganache

make transactions. It also provides access to the blocks, transactions, deployed contracts, events, and logs in the blockchain, which were useful for the implementation.

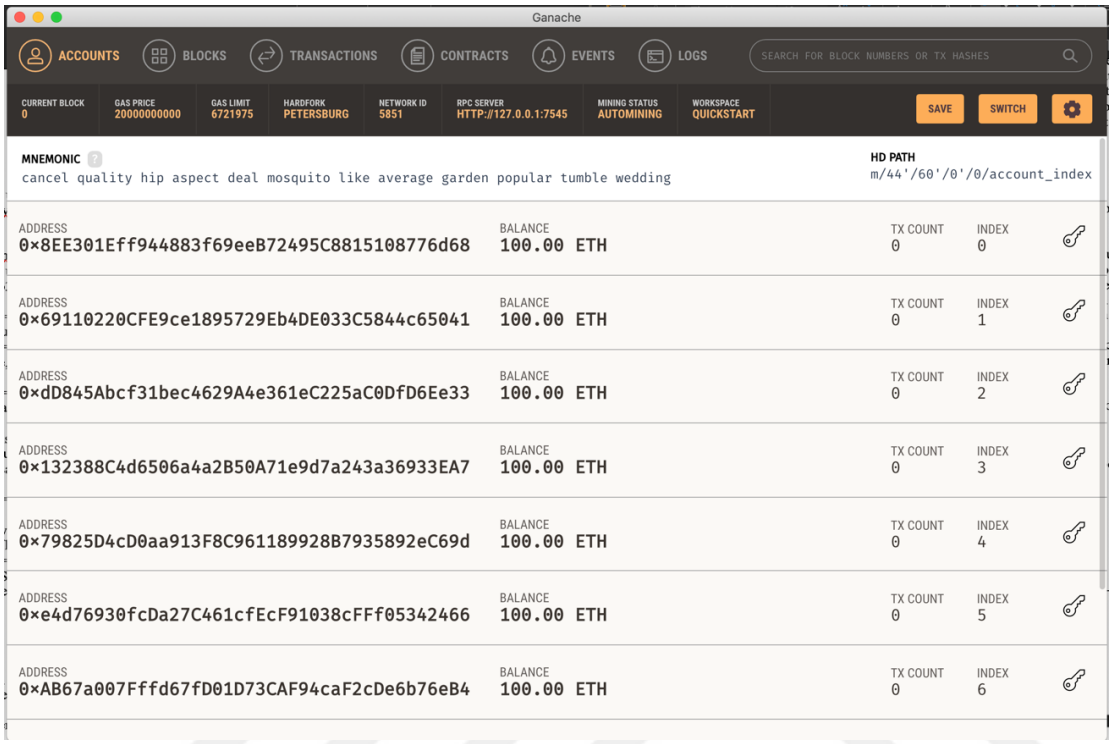

Figure 5.1: Ganache graphical user interface

Truffle [10] is a development framework for Ethereum projects. It increases the development speed by allowing writing, testing, and deploying smart contracts in to an Ethereum blockchain. It also includes a Solidity compiler, and gives access to contract ABIs, which are JavaScript interpretations to define smart contracts during implementation. Integrating front-end software to smart contracts is a complex task that requires many configurations. We used Truffle Boxes to start our project. Boxes are boilerplate starter projects built by Truffle or community, preparing a template of an empty project for the developer.

MetaMask is used for the connection to Ganache and it deals with account management and transaction signing for our PoC DApp.

WebStorm[11] is the JavaScript IDE that we used for developing front-end and smart contracts.

[10] https://trufflesuite.com/truffle

[11] https://jetbrains.com/webstorm

5.2. Swarm Usage

In order to upload and download encrypted content to Swarm, a user is required to be running a local Swarm node. Thus, users need to run local Swarm nodes in order to sell and buy content on our proposed DApp.

Swarm provides a CLI for interaction, but there is also a graphical user interface as a web service that runs on TCP port 8500. A user can check if the node is running, or download the content from Swarm, using web interface with a browser over 127.0.0.1:8500 (Figure 5.2).

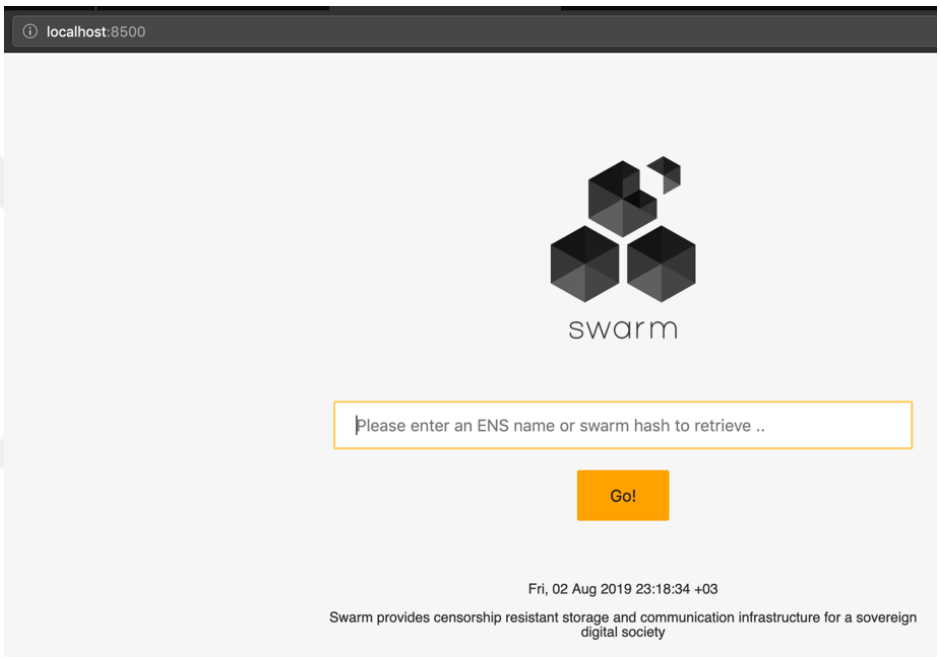

Figure 5.2: Swarm Web Interface

A seller can upload a content that she wants to be encrypted, with –encrypt method, which gives the reference address of the encrypted content.

```
serhat:swarm$ swarm up --encrypt content
88658c77457dc505fc17011c61ffa7b60594a587122cc24bfcd172b01efca7446927a9f5011e0e25167c4
d0962b058198983e430d4afdd83fa4595984cfaf7a7
```

After she uploads the encrypted content, then she wraps this encrypted content with a secret (secret key, password) she decides, which gives the reference of the encrypted content with the new password.

```
serhat:swarm$ swarm access new pass --password secret.txt 88658c77457dc505fc17011c61f
fa7b60594a587122cc24bfcd172b01efca7446927a9f5011e0e25167c4d0962b058198983e430d4afdd83
fa4595984cfaf7a7
```

Reference addresses of both the encrypted content and the encrypted secret password file are stored in the smart contract. Buyer decrypts the password file using PKC.

Reference address of the encrypted content and decrypted password are used by the buyer to download and decrypt the encrypted content from her local node.

```
serhat:swarm$ swarm --password secret.txt down bzz:/63db947d4a3d6cb605c1338b50964a667
4bce4fb88d0d443056b112a901c8256
serhat:swarm$
```

As can be seen below, this encrypted content cannot be accessed without a password or the secret file itself.

```
serhat:swarm$ swarm down bzz:/63db947d4a3d6cb605c1338b50964a6674bce4fb88d0d443056b112
a901c8256
Fatal: download: unauthorized
serhat:swarm$ swarm --password pass1.txt down bzz:/63db947d4a3d6cb605c1338b50964a6674
bce4fb88d0d443056b112a901c8256
Fatal: download: unauthorized
```

One of main reasons we chose Swarm over IPFS is, as mentioned earlier, Swarm presents built-in complex access control schemes.

5.3. Smart Contract Implementation

We implemented our Purchase contract according to the safe remote purchase via double escrow design described in Section 4.3. We used the source code given in Solidity documentation[133] as a base for this contract, then we adapted it to our overall design where we exchange digital content online instead of offline delivery of the goods. We also implemented Contract Management contract according to our design, so that sellers can deploy new Purchase contracts using this contract. Both of the contracts are written in Solidity version 0.5.x, latest version of the Solidity for security and adaptability considerations (see APPENDIX).

5.4. Front-end Implementation

As mentioned above, we used Truffle Boxes to start our project. For that purpose, we used chainskills-box as the truffle box, and chainlist as a basis for the frontend software, which are both in the chainskills[12] repository. The entry point to our DApp is the *index.html* file which sketches the GUI of our DApp. The frontend interaction with the backend was handled in *app.js*. In order to use web3.js for blockchain interaction, and to access active accounts in the browser, we needed to initialize web3 by using the web3 object injected by MetaMask or Ganache. Then we request user's permission to access account information.

[12] https://github.com/chainskills/

```
if (typeof web3 !== 'undefined') {
   // Legacy dapp browsers
   App.web3Provider = web3.currentProvider;
} else {
   // If no injected web3 instance is detected, fall back to Ganache
   App.web3Provider = new Web3.providers.HttpProvider(
      'http://localhost:7545');
}
web3 = new Web3(App.web3Provider);

// request account access permission
ethereum.enable()
   .then(function (accounts) {
   })
   .catch(function (error) {
      // Handle error. Likely the user rejected the login
      console.error(error)
   })
```

In order to access the components of the deployed smart contract, we need the ABI of the contract, as described in Section 3.3.1. Truffle automatically loads this ABI into a JSON file, so we initialized the contract with this ABI.

```
initContract: function () {
   $.getJSON('Purchase.json', function (purchaseArtifact) {
      // Get the necessary contract artifact file and instantiate it with truffle-contract
      App.contracts.Purchase = TruffleContract(purchaseArtifact);
      // set the provider for our contract
      App.contracts.Purchase.setProvider(App.web3Provider);
      // listen to events
      App.listenToEvents();
      // Use our contract to retrieve contents from the contract
      return App.reloadContents();
   });
```

DApp listens to the events of the blockchain and reloads the front-end automatically. First, an instance of the deployed contract is accessed then the Log function from the smart contract is called with JavaScript promises. Calls returns results, and this result is shown to the user. As an example, we listen the event triggered by buy content action below.

```
// listen to events triggered by the contract
listenToEvents: function () {
   App.contracts.Purchase.deployed().then(function (instance) {
      instance.LogBuyContent({}, {}).watch(function (error, event) {
         if (!error) {
            $("#events").append('<li class="list-group-item">' + event.args._buyer + '
```

```
bought ' + event.args._name + '</li>');
        } else {
            console.error(error);
        }
        App.reloadContents();
    });
```

Calling the functions of the smart contract is handled asynchronously. As an example, we show the implementation of the *buyContent* function below. This function acts as a middleman between the user interface and smart contracts. It gets inputs from the user interface, and then calls the *buyContent* function from the smart contract via a transaction, using the particular instance of the deployed contract. For data storage and retrieval of the encrypted content, secret, and buyer PGP public key, we used Swarm. This function uploads the public key of the buyer to the Swarm, and sends the returned reference address to the smart contract, together with the value that is sent as the deposit. For Swarm interaction, we utilized erebos.js based on file storage examples in its official documentation[13].

```
buyContent: async function () {
  event.preventDefault();

  // retrieve the content price
  var _price = 2 * parseFloat($(event.target).data('value'));
  const name = $('#buyerName').val();
  const email = $('#buyerEmail').val();
  const passPhrase = $('#buyerPassPhrase').val();

  var buyerKeyPair = await getPublicKey({name, email, passPhrase});
  const _buyerPublicKey = buyerKeyPair.publicKey;
  const _buyerPrivateKey = buyerKeyPair.privateKey;

  swarmClient.bzz
    .upload(_buyerPublicKey, {contentType: 'text/plain'})
    .then(hash => {
      $('#buyerPrivateKey').val(_buyerPrivateKey);
      downloadTextAsFile($('#contentName').text() + '.pk', _buyerPrivateKey);

      App.contracts.Purchase.deployed().then(function (instance) {
        return instance.buyContent(hash, {
          from: App.account,
          value: web3.toWei(_price, "ether"),
          gas: 500000
        });
      }).catch(function (error) {
```

[13] https://erebos.js.org/docs/examples-storage

```
            console.error(error);
        });
    })
},
```

For generating elliptic curve key pairs, and the encryption/decryption using PKC, we utilized OpenPGP.js in our app.js, based on the examples in its official documentation[134]. As an example of using PKC, we show the implementation of *downloadSecret* function below. This function decrypts PGP private key of the buyer with the passphrase that only buyer knows, downloads the encrypted secret from Swarm, decrypts the secret using buyer's private key, and downloads the decrypted secret in to local directory with the name of the content on sale.

```
downloadSecret: async function () {
    const passPhrase = $('#buyerPassPhrase2').val();
    const publicKeyHash = $('#buyerPublicKey').text();
    const secretHash = $('#secretHashAddress').text();
    const publicKey = await downloadFromHash(publicKeyHash);
    const encryptedContent = await downloadFromHash(secretHash);
    const privateKeyObj = openpgp.key.readArmored(this.buyerPrivateKey).keys[0];
    await privateKeyObj.decrypt(passPhrase);

    const optionsProvider = {
        message: openpgp.message.readArmored(encryptedContent),
        publicKeys: openpgp.key.readArmored(publicKey).keys,
        privateKeys: [privateKeyObj]
    };
    const decryptedContent = await
openpgp.decrypt(optionsProvider).then(decryptedMessage => {
        return decryptedMessage.data;
    });
    downloadTextAsFile($('#contentName').text() + '.txt', decryptedContent);
    console.log(decryptedContent);
},
```

In the smart contract, we enforced authorization for seller or buyer specific functions (see APPENDIX). If an invalid request is made by the user, the smart contract will throw an exception and break the transaction. In addition to this, as a second layer of security, and to protect users from invalid operations and spending gas for nothing, we did hide or show utility buttons according to the contract state and account address. So, a user is not able to make an invalid action through the user interface of the implemented DApp. Nevertheless, if these access restricted functions are called manually via a geth clients by a third peer, as we told above, the transaction will fail, and that peer will lose the spent gas.

5.5. A Test Scenario

In this section we test the proof of concept secure content sharing DApp we implemented to ensure that it meets the pre-defined functional requirements. For this purpose, we fulfill a scenario where a seller publishes a content and a buyer buys it with all the steps of these actions.

Before selling and buying a content, a user has to connect to an Ethereum network and manage her accounts via MetaMask. There are several Ethereum networks that can be connected to, however, we connect to Ganache, which is our test Ethereum network in this study. During implementation, we configured so that Ganache listens RPC requests on TCP port 7545. For sell and buy actions , we take actions for separate seller and buyer accounts by switching the accounts in MetaMask, both of which have different address and private key pair.

In order to increase the privacy, MetaMask runs with privacy mode enabled by default. If the user does not authorize the DApp, user's Ethereum address and related information is not revealed to the DApp. As a result of this, a user needs to first authorize our PoC Secure Content Exchanging DApp as shown in Figure 5.3.

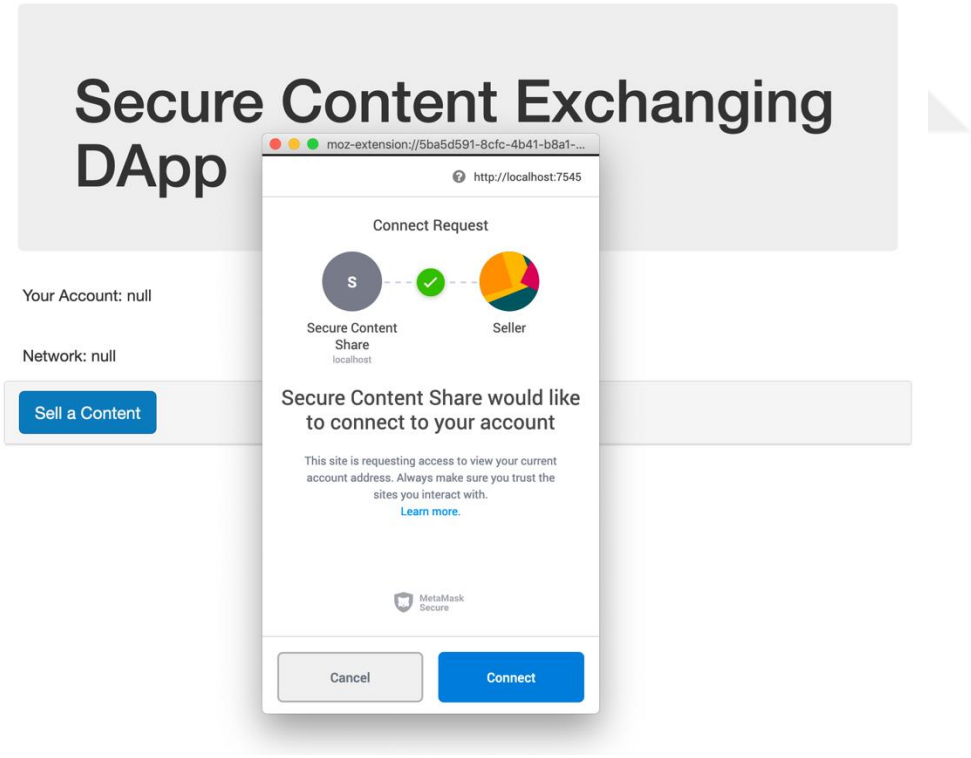

Figure 5.3: DApp MetaMask Authorization

After the permission is given to the DApp, the user can access the web interface of the DApp. Accordingly, account address, account balance, and connected Ethereum

network information are displayed in this interface. There is not any content on sale at the moment (Figure 5.4).

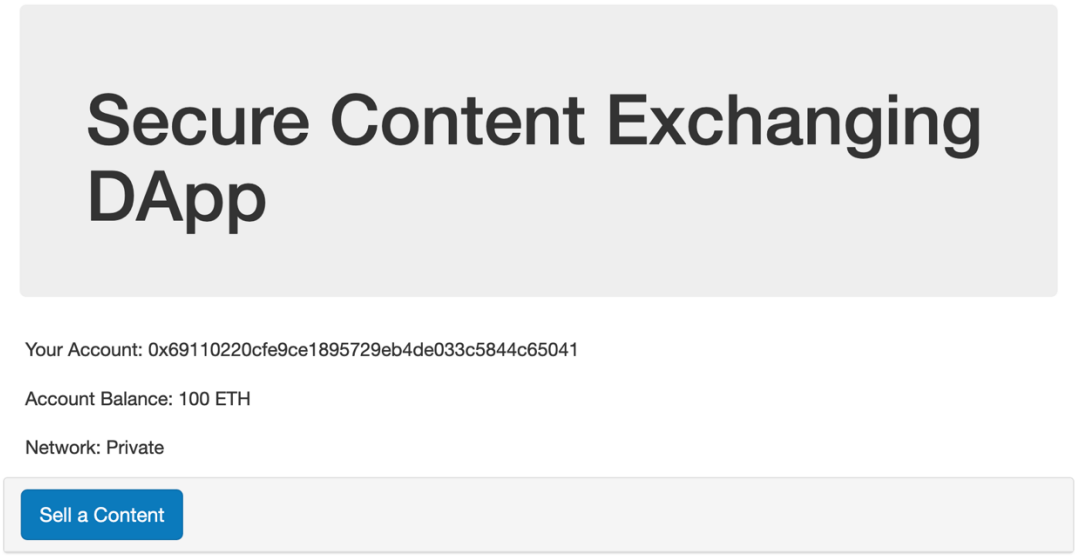

Figure 5.4: Content Sharing DApp initial interface

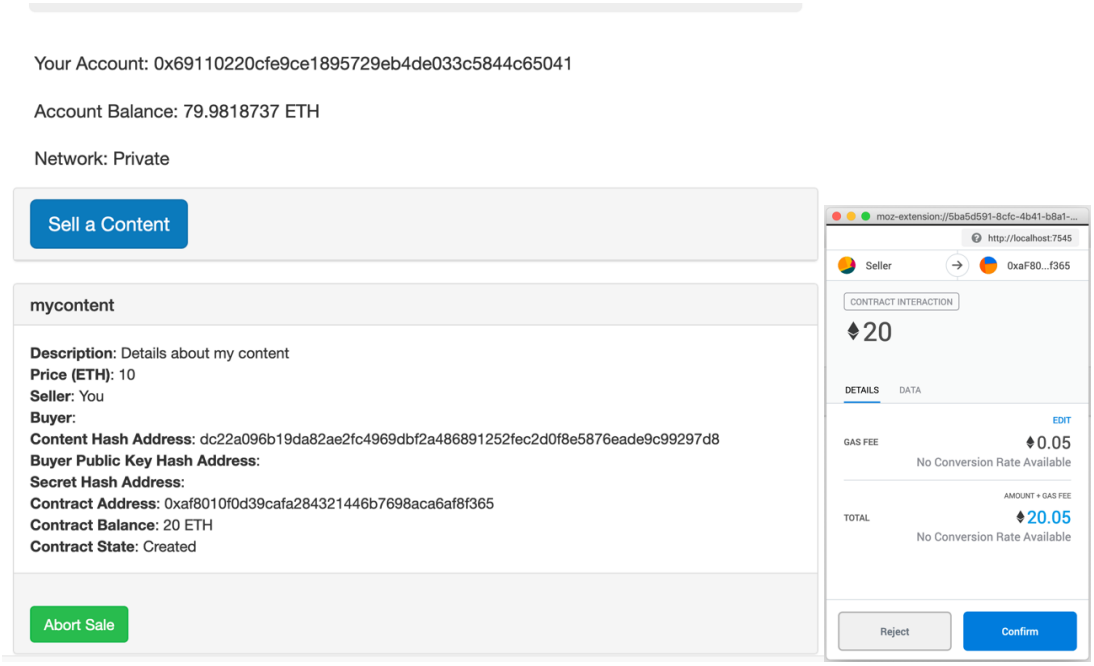

Figure 5.5: Put a content on sale in DApp

Put Content on Sale. We put a content on sale via *Sell a Content* button. The price of the content is set to 10 ETH, so after seller deposits 20 ETH, the content is now on sale as shown in Figure 5.5. As is seen, the buyer can abort the sale now. Also the details of the contract can be viewed. Note that the contract balance is 20 ETH, which is the deposit amount of the seller. Also, account balance of the seller is less than 80 ETH because of the transaction cost. The contract state is *Created*.

Buy Content. When we switch to the account that we want to buy the content with, we see that we can no longer abort the sale via user interface. Instead the buyer can generate OpenPGP key pair specific to this action by entering name, e-mail address, and a passphrase to encrypt and decrypt the private key (Figure 5.6).

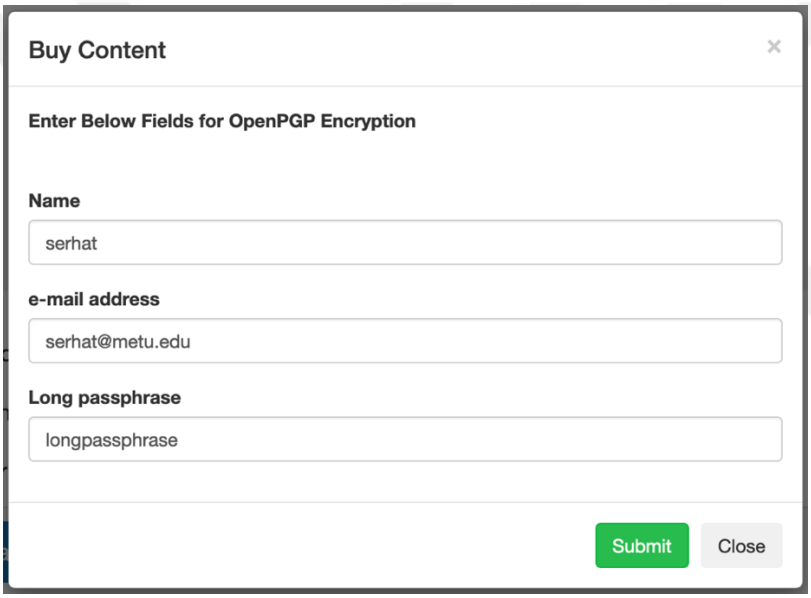

Figure 5.6: OpenPGP key pair generation

After the key pair is generated, buyer buys the digital content by transferring 20 ETH (10 ETH for the content and 10 ETH as the deposit) to the contract as shown in Figure 5.7. During the process, buyer's generated public key is uploaded to Swarm and its reference address is stored in the smart contract. Also, the encrypted private key is stored in buyer's local directory. When the transaction is signed and mined, state and other fields of the contract is updated.

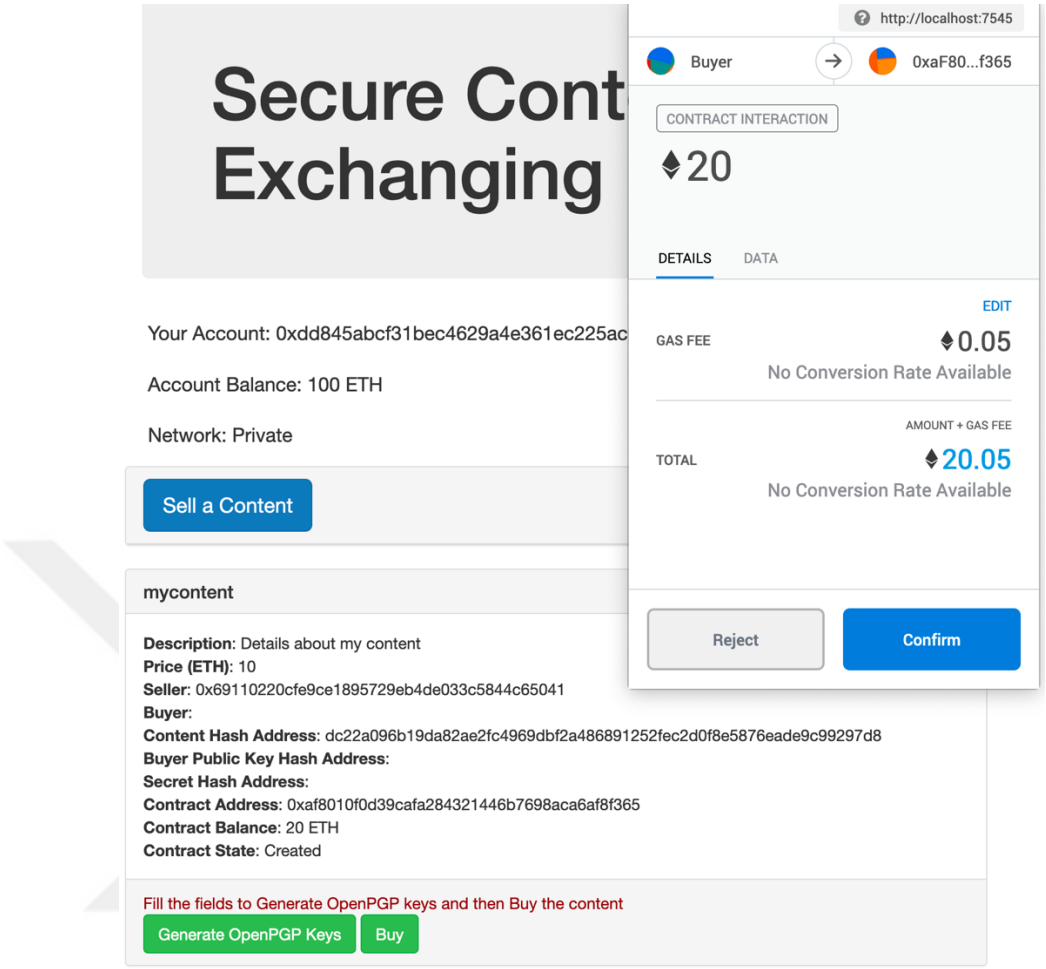

Figure 5.7: Buy a digital content

Deliver Content. When we switch back to seller account, we can observe that the contract state is *Locked*, and the contract balance is 40 ETH after the buyer deposits 20 ETH. The seller selects a file for encrypting, this file can be a very tiny sensitive information, or in this case it is the secret symmetric key to decrypt the large sensitive digital content. When the secret is delivered, we can assume that the content is delivered as it is already in the smart contract. This secret is encrypted with buyer's public key downloaded from Swarm, and the encrypted secret is uploaded to Swarm. This transaction does not transfer any Ether, but still some gas is spent to run the transaction (Figure 5.8).

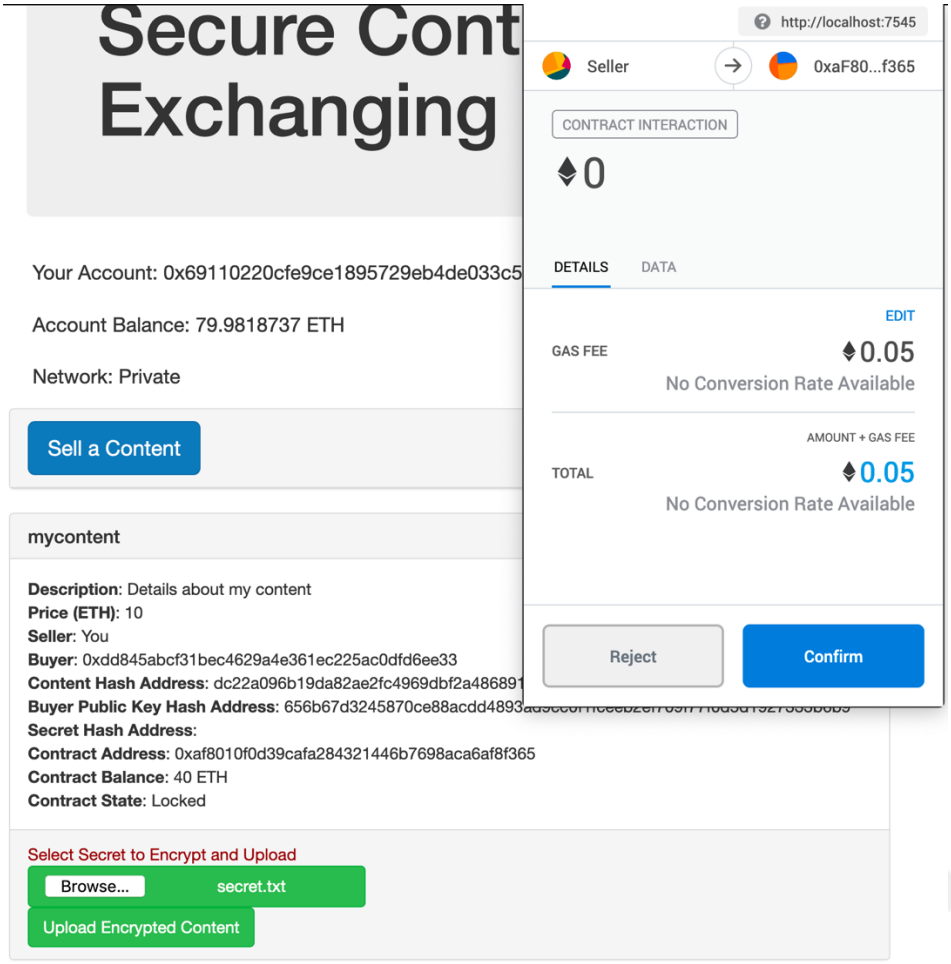

Figure 5.8: Deliver secret in DApp

Confirm Received. When we switch back to the buyer account, the buyer can download the encrypted secret and decrypt it with her private key decrypted with the passphrase. This is done via a call to the blockchain so no gas is spent. Then she confirms that she received the content by downloading the digital content from Swarm with the secret. This transaction does not transfer any Ether but some gas is spent (Figure 5.9).

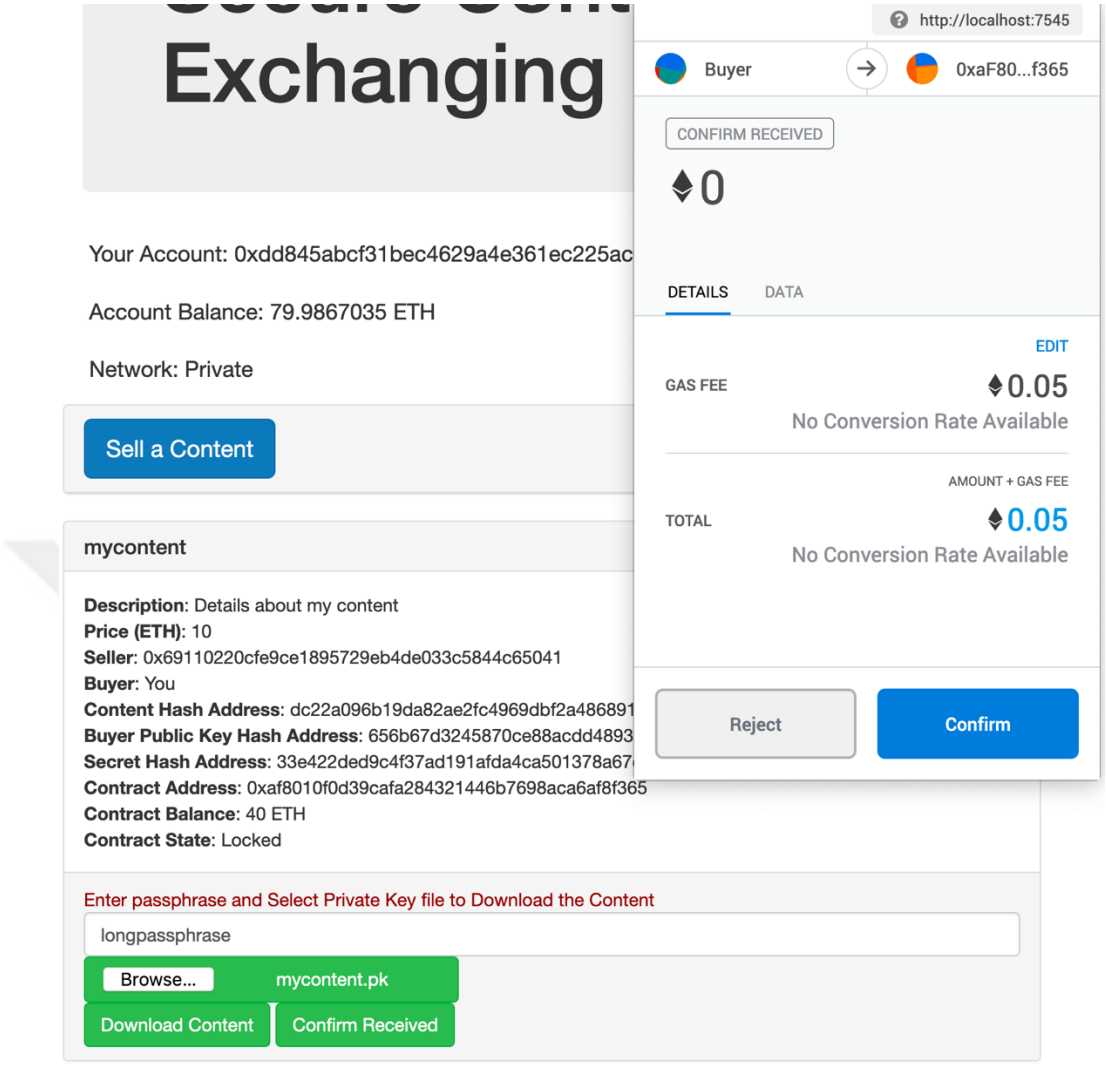

Figure 5.9: Confirm received in DApp

When the buyer confirms received, the exchange is finalized, and the escrow contract pays the deposits back to the seller and buyer; consequently the contract balance is 0 ETH, and the contract state is *Inactive* , as seen from a third user's Ethereum account (Figure 5.10).

When the exchange is finalized, addition the deposit, extra 10 Ether is transferred to the seller as the content price. The buyer receives her deposit after 10 Ether is deduces as the content price. Account balances of the users after the exchange is shown in Figure 5.11. As is seen, instead of 110 ETH and 90 ETH, seller and buyer has 109.97 ETH and 89,98 ETH, respectively. The remaining ETH was spent as gas during the exchange transactions. These transaction expenses are very high and as a matter of fact they are not accurate. The real amount of cost is way less than it is shown by Ganache. We thoroughly discussed the total costs for the usage of this DApp in Section 6.3.

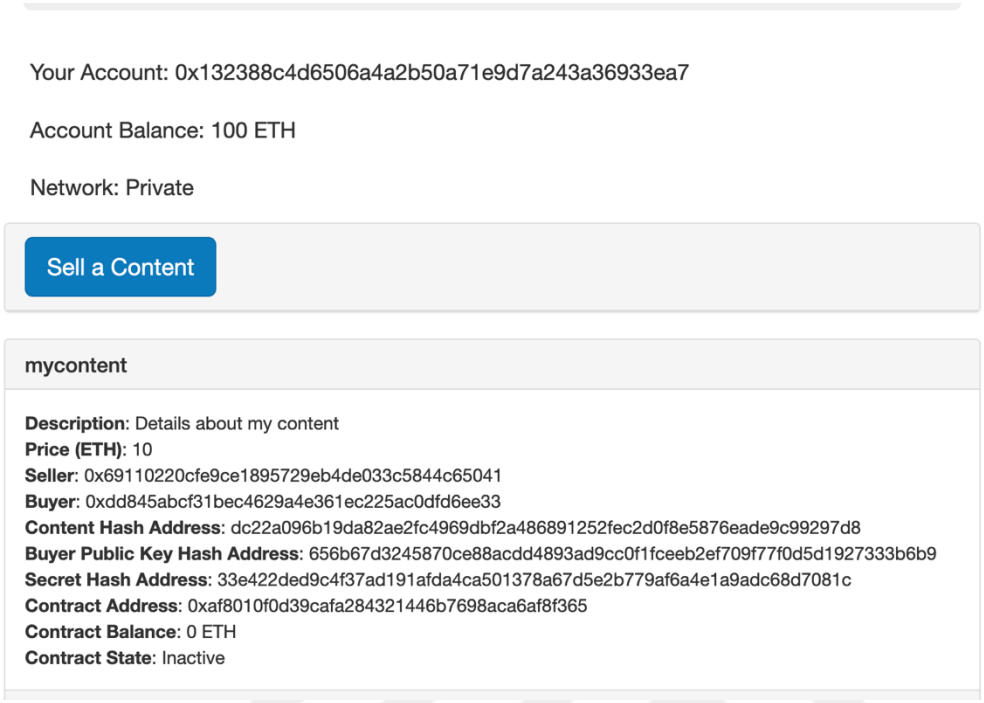

Figure 5.10: Exchange is finalized

Figure 5.11: Contract balances after the exchange

CHAPTER 6

EVALUATION AND DISCUSSION

In this chapter, we evaluate our proposed secure content sharing solution regarding the security goals of the study stated in the *Introduction* chapter, and all the non-functional requirements of the DApp stated in *Design* chapter. Accordingly, security of the system components, i.e., blockchain, Swarm, and smart contracts are explored, as well. In the meantime, we are also going to discuss further about these evaluations to have a broader point of view for the study.

6.1. Security

6.1.1. Smart Contract Security

Software vulnerabilities are the most common risks for an application. In this proof of concept, application logic is handled by smart contracts, and it is the most sensitive point for our DApp. Smart contracts in Ethereum intrinsically deal with Ether, thus actual money, so the security of the smart contracts is crucial. Accordingly, smart contract security analysis needs to be performed. Our smart contracts are written in Solidity language. This language is still not mature, and it is not mainstream compared to programming languages like Python, C, Java, etc., and as a matter of course there are not many academic studies related to security of the language, such as software verification. There have been several vulnerabilities in Ethereum smart contracts that originated from the Solidity programming language, EVM, and blockchain architectures. Note that these vulnerabilities are explored in Section 3.3.3.

Given the above, it is a common practice in the Ethereum community to use proven libraries and adopt security best practices. Accordingly, we based our Escrow logic on the official Ethereum source code examples. Nevertheless, we tested our smart contracts in Remix[14], a browser-based IDE, to find out if there is any vulnerability

[14] https://remix.ethereum.org

found. In addition to this, we performed security tests with SmartCheck[15], which analysis Solidity source code for the security vulnerabilities and best practices.

6.1.2. Private Key Security

In proposed scenario, account authentication is provided by MetaMask, acting as the identity vault for the interaction with the Ethereum blockchain and smart contracts. Transaction signing is done with the private key stored by MetaMask. Internet connected wallets, i.e., hot wallets, are naturally prone to attacks, this is why it should be ensured that the MetaMask account is kept safe, and the private key is not lost. Both MetaMask and OpenPGP stores the AES encrypted version of the private key, so we can consider that both ECDSA private key in MetaMask, and OpenPGP private key buyer stores are secure as long as passphrases used for encryption are not compromised.

For generation of the key pairs and signing transactions, Ethereum depends on PKC. More specifically, ECDSA, which utilizes *secp256k1* curve, is used in Ethereum [77]. ECDSA key pair is generated with this algorithm, and transactions are signed with ECDSA private keys and verified by related ECDSA public keys. An Ethereum account address is actually derived from calculating the Keccak256 hash of this ECDSA public key, and taking the rightmost 160 bit of the result [77]. Swarm also uses the same procedure described above, in order to integrate with Ethereum blockchain in the Ethereum ecosystem. As can be seen, overall security of the Ethereum and Swarm depends on the security of this secp256k1 curve and Keccak256 hash functions, which are proven to be secure since to this date. So, unless these cryptographic primitives are broken, we can consider the Ethereum and Swarm are secure.

6.1.3. Confidentiality

Confidentiality is the most critical non-functional requirement for our proof of concept implementation. As discussed before, Ethereum does not have a built-in privacy support for the data stored in its blockchain. Smart contract data and logic is also transparent. Consequently, to enable the information to only authorized users' access, we need to enforce off-chain confidentiality. For Alice and Bob to exchange secret information between each other without a pre-shared secret, they need PKC. In order to ensure confidentiality of the digital content, we utilized PKC for key generation and encryption of the secret. Accordingly, to implement cryptography in our proof of concept DApp, we used OpenPGP.js[16], which is a JavaScript implementation of the OpenPGP protocol. To provide security, we chose options that would make the encryption as strong as the modern cryptography standards require. A strong key generation is crucial and for this purpose we used elliptic-curve key generation and

[15] https://tool.smartdeck.net

[16] https://github.com/openpgpjs/openpgpjs

selected *secp256k1* curve. We chose this curve in order to correspond to Ethereum Web 3.0 stack ECC security.

For key generation, we could have used RSA or Elliptic-Curve Cryptography (ECC) as the PKC primitive for the PGP key pair of the OpenPGP protocol. RSA requires much larger key sizes to achieve sufficient security level compared to ECC. For instance, for 128-bit security: RSA requires 3072-bit key, whereas ECC requires 256-bit key; for 256-bit security: RSA requires 15360-bit key, whereas ECC requires 512-bit key [135]. Even though public key operations are relatively faster with RSA, given the same key size, we opted for elliptic-curve key generation in our PoC to achieve a better security to computational cost compromise. PKC is far slower than symmetric key cryptography for encryption/decryption tasks. As a result of this, PKC is not suitable to encrypt/decrypt large files. Accordingly, we used PKC only for encrypting the symmetric key, which is used for encrypting/decrypting the digital content with a much faster symmetric key algorithm, i.e., AES.

In PKC, trust is the essential aspect of the scheme. Alice needs to trust that the public key, which is claimed to be Bob's, is actually Bob's public key, thus it is authentic. Which means the public key is not tampered by Chuck. In modern internet security, such as in TLS, this is accomplished by using digital certificates issued by trusted Certificate Authorities. If so, in our scheme, how does the seller make sure the received public key is authentic? There are two aspects of this: First, a man-in-the-middle attack is not possible in our scenario, because the public key is never transferred over an unsecure HTTP traffic. It is uploaded to the decentralized Swarm network, through the local node running in the buyer's computer. As mentioned in Section 3.5.3, data is distributed and stored in chunks, and as a result, a reference to the file in Swarm is created. This reference address generation is deterministic, so one particular public key file can only have one reference address, and once it is created it cannot be changed. If you put a different public key in Swarm, it would be accessible from a different reference. This would assure the integrity of the file in Swarm network. Also, when a Swarm address is requested, the data and the requested address are compared, which makes a man-in-the-middle attack infeasible. Second, derived Swarm reference is written to the smart contract with a transaction signed by seller herself. Since this address is signed by the seller with her Ethereum account private key, we can conclude that this Swarm reference information is correct, thus authenticity of the public key is assured. These two aspects suggest that the public key information accessible from the reference address that is stored in the blockchain is trustable.

6.1.4. Integrity

In our proposed scheme Ethereum blockchain acts as the distributed ledger that immutably stores transactions and manages cash transfer between buyers and sellers. On the other hand, Swarm acts as the decentralized storage with immutable content addressing. Immutability of the both blockchain data and Swarm contents is a key aspect of the overall integrity of the system.

Ethereum, as it is a permissionless blockchain, makes it possible for separate users that do not trust each other to securely share valuable data with making tampering

impossible. This tamperproof architecture is enabled by linking the blocks with the hash of the previous block and storing a Merkle root hash of the transactions and the state of the blockchain in the block header. This is ensured with the proof-of-work consensus mechanism. Both of these processes are dependent to the decentralization of the network. If either of this process is compromised, the security of the whole system is compromised. The 51% vulnerability, Sybil attacks, and selfish mining are key risks for the permissionless blockchain systems, regarding the level of decentralization of the system. There is another risk with decentralization, that is, the possibility of the power is significantly biased in favor to a few nodes, called mining pools. Note that the issues we discussed above are thoroughly explored in Section 3.1.2 and Section 3.1.5.

Cryptographic hash functions are used in both Ethereum blockchain and Swarm in order to provide the immutability of the data, thus ensuring the integrity of the system. In blockchain, these hash functions are used during the processes of proof-of-work consensus, chaining the blocks with the hash of the previous block, and Merkle root hash generation. In Swarm, hash functions are used to generate reference addresses of the contents. The contents are mapped in to a Merkle tree, buy packaging the reference address of the chunks in to another chunk with its own hash address [114]. Both Ethereum and Swarm uses Keccak256[17] hashing algorithm, and as mentioned above, unless this algorithm is broken, we can consider the integrity of Ethereum and Swarm is assured.

In the front-end of the DApp, interaction with the Ethereum network is handled locally via MetaMask. Similarly, interaction with the Swarm network is done over a local node. As a result of this, the data is never sent to the internet over an unsecured network. Thus, as long as the integrity of the Ethereum and Swarm is assured, we can conclude that our DApp provides integrity.

6.1.5. Availability

The proposed DApp is utilizing the blockchain and Swarm, both of which are operating on peer-to-peer networks with no central point of failure. Due to this architecture, sufficient redundancy of the data is achieved. Note that both Ethereum and Swarm are relatively new technologies, the software in decentralized architectures and the community are continuously evolving. We can assume that, as long as Ethereum and Swarm peers continue to exist, the proposed architecture will be available.

6.1.6. Access Control

To evaluate further, we can discuss about access control and access revocation aspects regarding the shared sensitive content. As mentioned before, there is a single key, i.e., buyer's public key, that enables encryption of the content, thus giving access to the

[17] https://keccak.team/keccak.html

buyer. We only have seller and buyer roles in our context. Therefore, there is no need for fine grained access control in our proposed scheme. On the other hand, when a file is uploaded to Swarm the file is propagated to other nodes as well. Thus, the publisher loses control of the file when she uploads it as there is no delete mechanism in Swarm. As a result of this, once the access is granted, there is no access revocation mechanism for the content.

6.2. Privacy

As described in Section 3.4, Ethereum blockchain is actually pseudonymous [103], and these pseudonyms can be deanonymized [21] [104] [105]. Thus, true anonymity is quite hard to achieve in blockchains. Nevertheless, compared to centralized exchange platforms where many personally identifiable information, such as name, e-mail address, mail address, credit card information, etc., is actually required to use the service; better privacy is achieved in our proposed scheme, as just an Ethereum account and private key is enough to identify and authenticate a user.

In its older versions, MetaMask used to reveal Ethereum addresses of the user to the website the user browses. This used to occur because MetaMask used to automatically inject a web3 object to interact with the webpage. As Ethereum addresses are unique to users, exposing of it to every webpage yields privacy concerns. In an attempt to solve this, MetaMask has offered a privacy mode in a recent version, which is enabled by default. With privacy mode enabled, a DApp needs to ask for permission, and users can choose to use the DApp and reveal their Ethereum addresses to the website by granting authorization. This is a significant step to protect user privacy in decentralized applications.

For some particular services, more personal information can be required from users. In order to give users the control of the information they share with the third parties, there are decentralized identity management systems, such as uPort [18], which corresponds with the decentralized nature of our DApp. So, a better privacy could be still achieved with these systems, as the personal information cannot be shared with a third party, without the approval of the user, as the user is the only one in control with her own information. Since this is a proof of concept design, we did not utilize any decentralized identity management systems, but it can be considered in a production-ready DApp.

6.3. Costs

The implemented DApp does not need any TTP to function, as it is data, transactions, and logic is operating on decentralized systems. A decentralized consensus mechanism

[18] https://uport.me

is necessary to make this possible. However, this consensus mechanism, i.e., PoW, requires miners to allocate hardware resources in order to verify transactions. Thus, when a smart contract is executed in Ethereum and if the transaction changes the state of the blockchain, triggered operations of the transactions are executed in every node, which requires hardware resources. Allocating these resources is paid with gas by the transaction senders, so every action that changes the global state of the Ethereum costs gas. In short, a buyer and seller need to pay gas fee for their transactions in our proposed scheme. In order for this DApp to be feasible, the cost for selling and buying actions needs to be reasonable.

The gas fees for EVM operations used in the cost calculations of this thesis were derived from Gavin Wood's Ethereum Yellow Paper [77], and the current metrics for the Ethereum gas market was derived from ETH Gas Station [136], as depicted in Figure 6.1. The gas price is dynamic and changes depending on transaction senders and miners. If a user wants to have a faster transaction, she can increase the gas price. As the gas price for following calculations, we take the standard gas price, which is 3 Gwei (Figure 6.1). Also, we took 1 ETH as around 225 USD, the actual price taken on the same day with the above metrics (August 2019). This price is dynamic as well, so overall costs of operations in USD are actually dependent to gas price and ETH price at the time of the calculation.

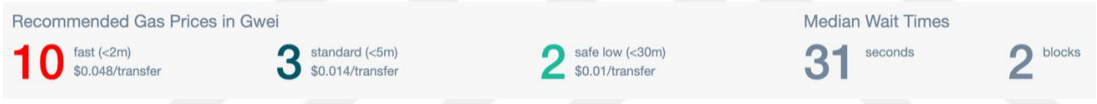

Figure 6.1: Ethereum Gas market metrics

Cost for a transaction that creates a contract is 32000 gas. So a seller has to pay 32000 gas for each sale. For each transaction, 21000 gas is required, so both buyer and sellers pay this amount and plus the costs for other operations, i.e., SSTORE operation for storing data permanently in the blockchain. The word size of the EVM machine is 256 bits, and to store a new non-zero 256-bit word in smart contract requires 20000 gas. Speaking of which, changing the value of a previously stored word requires 5000 gas. When we check the Ethereum Yellow Paper for gas prices, we can see that SSTORE operation is the most expensive operation for dealing with the data in a smart contract.

If storing a 32-byte word costs 20000 gas, then it means storing a 1KB file costs 640,000 gas ((1024 x 20000) / 32), and storing a 1MB file costs 655,360,000 gas (640,000 x 1024). The further costs for these operations in Ether and USD are given in Table 6.1. As is seen, storing 1 MB data or 1 GB data in Ethereum costs 444 USD and 454,656 USD, respectively! In fact, there is a block gas limit in Ethereum which is set to keep the transaction time required by each block at minimum. Block gas limit is around 6,700,000 gas, so technically you cannot request an operation to spend more than this amount per block. Nevertheless, theoretically, these large data can be stored in fragments in separate blocks. Given these numbers, Ethereum blockchain is not suitable for storing large files, indeed.

Table 6.1: Costs of Ethereum operations in ETH and USD

Operation\Cost	**gas**	**ETH**	**USD**
Transaction	21000	0.000063	0.01418
Create Contract	32000	0.000096	0.0216
Store word	20,000	0.00006	0.0135
Store 1 KB	640,000	0.00192	0.4339
Store 1 MB	655,360,000	1.966	444
Store 1 GB	671,088,640,000	2013	454,656

In order to make this DApp usable, we upload large files to Swarm and store the reference addresses in the smart contract. Specifically, digital content, secret file, and buyer's OpenPGP public key were stored in Swarm. Swarm reference addresses are 32 bytes which are compatible with 32-bit word size of the EVM. Thus, each of this operation costs 0.0135 USD.

As we did not deploy our smart contracts in the Mainnet, in order to estimate the total amount of transactions costs that buyer and seller would spend using our proposed DApp, we used the gas estimation functionalities of Ganache and Remix. These estimations are not precisely accurate so we wanted to see if the transactions costs in both of the tools are close to each other, so that we could come up with a more accurate estimation. First, we checked the remaining Ether balances of the users after a sale of a content was finalized, in which all the transactions were made with standard gas prices. Since Ganache gas prices is overwritten by web3 default gas price, we could not set the gas price to 3 Gwei, instead it was taken as 100 Gwei, which explains the high amount of costs we saw before in Figure 5.11. When we count the gas price as 3 Gwei, total costs for the seller and buyer drops to more reasonable amounts. We compared transactions costs for each action both in Ganache and Remix and we confirmed that the amounts are close to each other.

The implemented DApp has five use cases as described in Section 4.1. The estimated transaction costs and their Ether and USD values for each of these five actions are given in Table 6.2. As can be seen, putting a content on sale is by far the most expensive action for users. In this action there is base transaction cost, contract creation cost, additional cost for the data fields, and execution costs. Depending on the cost estimation, we can deduce that selling a content in the proposed DApp costs roughly 1 USD to the seller. On the other hand, cost of buying a content to the buyer is less than 0.1 USD. We assume a cost which is less than 1 USD can be considered acceptable for many of the digital content exchanges, considering the loss of money to the TTPs in the centralized systems.

Note that, there is also a cost for deploying the *ContractManagement* contract, which is 2892138 gas, 0.00867 ETH, and 1,675 USD. This is a one-time payment paid by the DApp *owner*. Nevertheless, this owner has no privilege in the system, and cannot influence any of the deployed Escrow contracts, unless she is either the seller or the buyer.

Table 6.2: Costs of actions of the DApp in ETH and USD

User	**Action**	**Transaction Cost (Gas)**	**ETH**	**USD**
Seller	Put Content on Sale	1796171	0.0053	1,04
Seller	Abort Sale	46776	0.0001403	0,027
Buyer	Buy Content	132852	0.00039	0,077
Seller	Deliver Content	92042	0.0002761	0,053
Buyer	Confirm Received	46460	0.0002761	0,027

From the users' point of view, we can deduce that, the more expensive the content is, the less significant buy and sell costs become. This is because the cost is fixed and not affected by the price of the content. In conventional centralized exchange platforms, a seller generally waits at least a week to receive the money the buyer paid for the content. The TTP, i.e., the website, generally cuts share from seller, as well. As a result of this seller might put the price more than she intended in the first place, and the buyer pays more than she actually supposed to. As long as this cut is more than our transaction costs, which is less than 1 USD, using our proof-of-concept is actually cheaper to use, compared to conventional exchange platforms.

According to Solidity documentation [94], *string* should be used for arbitrary-length, and *bytes* should be used for data with limited length. This is because *bytes* is cheaper then *string* in gas consumption. We stored Swarm reference address values in *string* type, rather than *byte32* type for simplicity. Also, limiting the string sizes for name and description of the content fields is necessary to keep the transaction cost reasonable. If the user is required to enter very long string, this field can be stored as a Swarm reference as well. Although there may be other ways to reduce the gas consumption, our DApp is a proof-of-concept so more optimizations can be performed in the future.

6.4. Scalability

The scalability of our the PoC DApp is inherently bounded with Ethereum blockchain scalability. Currently Ethereum process less than 20 transactions per second. Current transaction capacity of Ethereum is relatively low compared to several other programmable blockchain networks. Ethereum uses PoW as the consensus mechanism and each node in the network is theoretically equal (although this is not valid in the real world as there are mining pools and each node has different processing power). Ethereum will switch to a variant of PoS consensus in its next major update, which is also used in other blockchain networks with higher transaction capacity. Although PoW theoretically provides more decentralization, it comes with the cost of scalability to ensure the security of the blockchain. Main reason behind the scalability bottleneck in Ethereum is because all the operations performed in EVM are actually performed in each node. The scalability issue in Ethereum is planning to be solved by a process,

called sharding, in the new major upgrade of the Ethereum. Note that, sharding is discussed in detail in Section 3.1.4.

Scalability, decentralization, and security of a blockchain are all dependent on each other, and it is a challenging task to achieve all of them at the same time with sufficient optimization. Buterin [137] called this problem as the *scalability trilemma*. He claims that it is hard to achieve all three of these features in a blockchain, and it is a problem that should be solved. Figure 6.2 illustrates this trilemma, and for instance, when a blockchain is secure and decentralized, it is hard to make it scalable, and so on. Many blockchains decrease decentralization level to achieve scalability.

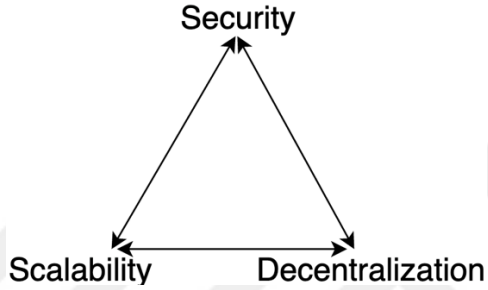

Figure 6.2: Scalability Trilemma in Blockchain

6.5. Usability

From the end user point of view, a DApp is not very different compared to ordinary web applications. However, a DApp still requires a DApp browser, which, in this case is MetaMask. In our prosed scheme, a user also needs to run a local Swarm node. Ethereum technologies and peer-to-peer networks are still not common, and most of their users are computer savvy people. So, the usability of this DApp for an ordinary end-user is not going to be as high as an ordinary web application in the near future, not before these technologies become mainstream. There are projects to decrease the steps necessary for a user to use DApps, such as *Universal Login*[19] which aims to solve the adaptation barrier issue. In addition to these, if the DApp is hosted in Swarm, the website will be accessible by a 64-character long address, which is not practical to use. Hence, using decentralized name services, i.e., ENS, would increase the usability of the application.

From the security and usability tradeoff point of view, we tried to keep the DApp as usable as possible, without compromising the security of it. Generating OpenPGP key pairs in Swarm CLI, encrypting the secret and decrypting it all through CLI would actually make the DApp more secure from the end user's point of view. However, this

[19] https://universallogin.io

will decrease the usability of the system. All in all, this is a proof of concept design, and a better security and usability balance can be achieved in the future.

6.6. Performance

A user's interaction speed with the frontend of a basic DApp is not different compared to other web applications. However, in our proof of concept, we have two actions that can affect the interaction performance: (i) interacting with Swarm, and (ii) PKC operations such as ECC key pair generation, and encryption/decryption of the secret. According to our tests, these operations does not cause a significant wait time for the user. Nevertheless, all the DApps whose smart contracts are deployed in public Ethereum mainnet require some wait time when the state of the smart contract is changed. This is the time for the transactions to be verified and mined in to the new block. According to ETH Gas Station [136] as shown in Figure 6.1, median wait time for a transaction is 31 seconds. Ethereum block time is about 15 seconds, thus this makes mining time of 2 blocks. It is most likely that a transaction with the standard gas price is executed in less than 5 minutes.

In addition to this, in order to make sure to prevent a double spending attack and ensure non-reversion of the blockchain, a certain amount of confirmations is necessary for the user to make sure that particular block is actually a part of the eventual consensus. It is required to wait for 10 confirmations to achieve a 99,99% probability of security, for blockchains with 17 seconds block time [138]. So, in Ethereum, in order to be in the absolutely safe side, three minutes (31 seconds for median wait time + 150 seconds for confirmation time) is necessary for a transaction with *enough* gas price. Considering the privacy protecting scheme of decentralized systems, this 3-minute delay can be considered acceptable. Three minutes is actually the fastest wait time for a safe transaction. For the standard gas price, this time would increase according to the congestion of the Ethereum network.

$$Fastest\ total\ wait\ time\ for\ a\ safe\ transaction = 180\ seconds$$

In conventional exchange systems the payment can be completed in seconds if the user identity or the credit card is verified. However, for purchases with anonymous identity or for new registrars, most of the time the credit card needs to be confirmed by the web application before the payment is issued. For international purchases sometimes the verification process takes more than hours. Required wait time is even longer for very high prices, because of the banking system anti-fraud restrictions. On the other hand, buying Ether is not a very straightforward process either, for a regular end user. Currently, a user can buy an Ether via exchange platforms, most of which require identification and credit card verification, as well. So, as a matter of fact, for a first-time user of an exchange platform, onboarding time requires more pre-work for decentralized applications, as most of the people are already adapted to the current financial system.

CHAPTER 7

CONCLUSION AND FUTURE DIRECTIONS

In this thesis, we have proposed and implemented a blockchain and smart contract based proof of concept in an attempt to study a secure digital content sharing platform built on a decentralized architecture. Throughout the thesis, our research demonstrates that secure content sharing between two parties is viable over a permissionless blockchain, i.e., Ethereum. Our approach enables separate users that do not trust each other to use the implemented decentralized application without the need for a TTP. Nevertheless, extra measures were taken to assure the confidentiality of the content, decentralized storage, and fair exchange between parties.

According to our validation and tests, the implemented design satisfies the functional requirements that were elicited during the design phase. The proposed architecture allows separate parties to use the system to securely exchange digital content. Via this DApp, a *seller* can put a digital content on sale, abort the sale if it is not bought by anybody, encrypt and upload a secret, and get paid when the content delivery is confirmed. On the other hand, a *buyer* can view the sales, buy and make payment for a content if the sale is open, decrypt and download the secret, confirm received and recover their deposit.

Nevertheless, for our implementation to be feasible, it has to satisfy certain non-functional requirements as well, which are given in the *Introduction* chapter as a complementary goal of the study, and others elicited during the design. Upon evaluating of the implementation, we confirmed that the proof of concept ensures the security of the sensitive data by providing the confidentiality, integrity, and availability of it. Certain amount of user privacy is provided as there is no required personal information to use the platform. We also confirmed that the built application incurs a reasonable amount of usage cost, provides sufficient performance, allows scalability, and it is actually relatively easy to use, thus our implementation is viable and usable indeed.

During this study, all of the research questions of the thesis, which are introduced in the *Introduction* chapter, are answered:

- *Is it possible to make a secure content exchange between separate parties with payment or contract, without the need for a TTP?*

Blockchains are suitable when it is required to eliminate TTP for the interaction of separate users while also providing a framework to make payment. This is achieved by the decentralized consensus schemes, which is PoW for Ethereum. In order to implement the application logic for content exchange, smart contracts were used without sacrificing decentralized architecture of the system.

- *How can we combine blockchain and storage of large data in a fully decentralized way?*

  As we have demonstrated in this work, storing large data in Ethereum is impractically expensive. Therefore we utilized Swarm, a decentralized storage system, native to Ethereum ecosystem. All the contents and other necessary large data was stored in Swarm. In order to ensure the immutability of the data, 32-bytes reference address of the data was stored in the blockchain.

- *How can we securely share content on a fully decentralized software and network architecture in a permissionless blockchain?*

  Security of the system is a key target for our study. Although Ethereum assure immutability, availability, auditability, reliability of the system, it does not provide privacy of the blockchain data. This is because all the transactions and smart contracts are accessible by any node. As a result, confidentiality of the data is not addressed in Ethereum. In the context of a digital content exchange, the content should be accessible by only the buyer. In order to provide this access control scheme, we employed a PKC encryption framework over OpenPGP protocol using elliptic-curve key pairs. In our design, the seller encrypts the secret with the public key of the buyer, thus the content, which is encrypted with a symmetric key, is decryptable and accessible by only the buyer.

- *How can we ensure fair exchange between separated users that do not trust each other, without a conventional TTP?*

  Protecting the interest of both buyer and seller is a challenge in decentralized systems known as the fair exchange problem. This problem is conventionally solved by depending on TTPs. Nevertheless, instead of a conventional TTP, we used smart contracts to enable remote purchase logic, which acts as the *trustless* mediator between users. With the intention of achieving a model of fairness, we applied a double escrow service in our smart contract design, by which both seller and buyer make deposits, so they cannot act maliciously. In our model of fairness, we ensure that a rational malicious party cannot gain any advantage over the other. By not depending on a TTP, our two-party fair exchange protocol avoids certain problems caused by it.

Note that, although the proposed DApp provides a framework for selling and buying a digital content in a secure way, it also allows the content owner to set the price to zero. In this case, DApp acts as the means for publishing digital content if the content is not encrypted. Also, it provides a platform for exchanging information and content

between any two parties, with end-to-end encryption. With a small amount of cost, users can send and receive secret information totally on distributed networks, through a secure, reliable, censor-free decentralized platform.

7.1. Contribution

In this thesis, we designed and implemented a decentralized platform that enables separated parties to securely exchange, buy and sell digital content by utilizing the Ethereum ecosystem. We used and explored Swarm as the decentralized storage system in our DApp, even though all the papers and thesis we investigated preferred IPFS. Furthermore, by applying PKC, this proof of concept enables users to exchange digital content without the need for any pre-shared secret. Also, a double escrow service is integrated via smart contract which makes it possible to achieve certain level of fairness between buyer and seller. Although there are related studies that explores these concepts, as far as we know, this is the only work that combines Ethereum blockchain, smart contracts with double escrow functionality as a measure of reasonable fairness in the context of *fair exchange*, PKC with *forward secrecy to mitigate data privacy issues in Ethereum*, decentralized storage to achieve a secure remote purchase over a fully decentralized architecture, decentralized application with a frontend for usability, and a detailed evaluation of the technology regarding its aspects such as security, privacy, scalability, cost, usability, and performance. We thoroughly explored Ethereum blockchain and its Web 3.0 stack; we discussed security, privacy, decentralization, scalability, and usage cost of the technology in detail. As a whole, this study brings many aspects of decentralization and Ethereum ecosystem in one place, hopefully as a comprehensive guide to the technology, and its application, specifically in digital content exchange.

7.2. Limitations and Future Work

Blockchain and smart contracts are very new technologies, so all of the back-end technologies we used in this study, such as Ethereum blockchain, smart contracts, and Swarm, are very new software. As a part of the Ethereum ecosystem, they are either beta versions, or proof of concept applications. The community is very vibrant, and the software changes often as it gets more mature with the feedback from earlier versions. As a result of this, all different software we used during this thesis is awaiting new major updates that would change or break the way it works in the current versions.

During this study, we designed and implemented a proof of concept to explore the viability of the Ethereum ecosystem for securely exchanging digital content. Although the developed software indicates that we achieved the main objective of the thesis, it is not production ready. As it is a proof of concept design, it has several limitations that can be addressed in the future.

- As the Ethereum blockchain, we used Ganache, which is an in-memory private blockchain for development purposes. Although Ganache imitates almost all

of the functionality of the Mainnet, we can also deploy our smart contracts to a public Ethereum test network or the Ethereum mainnet.

- Although we utilized Swarm to decrease the costs of data storage in our smart contracts, and actually decreased the cost to a reasonable amount, there are still space for improvement to have a more cost effective DApp.

- Although we use Swarm for content upload and download operations, we did not fully benefit from the access control functionalities of it. Swarm can provide access control mechanisms via Swarm account public keys, without the need for a separate key generation. As Swarm is fundamentally developed to integrate with the Ethereum blockchain, we can also integrate it to our DApp profoundly, though Swarm is still very immature.

- We implemented the smart contracts for multiple buy and sell operations, and multiple sale and buy operations can be made through a geth client. However, our front-end software supports only one buy and sell operation. Since the scalability of our DApp is totally dependent on the scalability of the Ethereum blockchain, and as this is a proof of concept design, we assumed that a single buy and sell set of operations is enough to illustrate how our design works in order to achieve the goals of this study. Nevertheless, front-end software can be further developed before using this DApp in Ethereum mainnet and hosting it on Swarm.

- As a key risk of using blockchain technology and decentralized storage systems, this proof of concept enables anonymous usage without censorship, which might encourage criminal activities.

7.3. Future Research

In this work, user authentication is handled by MetaMask and there is no identity management. This approach is enough to provide a platform for anonymously exchanging content, however, in order to have a more business-oriented product, identity management should be addressed. For the continuity of our decentralized architecture and provide privacy, a decentralized identity management should be utilized. Decentralized identity management is an important concept, and can be researched further along with its compatibility with the GDPR.

In order to be able to use our PoC DApp, there are many steps a user has to go through. This necessity decreases the usability of our DAPP, and it is actually a common problem with all Ethereum based DApps. There are studies to solve this problem and make the adoption process easier. *Universal Login* is an example project created for this purpose, and this topic can be researched in the future.

In order to solve the privacy issue of the Ethereum blockchain data, we applied PKC to assure confidentiality. Nevertheless, there are other approaches to solve this problem. Enigma, Hawk, Lightstreams, Parity secret store are example projects that

are built in an attempt to solve this privacy issue of permissionless blockchains. Using different cryptographic approaches, such as zero proof knowledge cryptography, these projects are aiming to enable sharing of a secret over permissionless blockchains. Ethereum is also going to implement zk-SNARKS in its next major update (aka Ethereum 2.0) for this objective. These projects and the protocols they provide can be researched further.

Fair exchange in two-party computations is a well-studied subject, however exchanging digital goods over blockchain and utilizing smart contracts is a new area of subject. In this thesis, our approach of fairness by incentivizing the parties with deposits via the double escrow smart contract, achieves fairness for rational users. Although unlikely, it is possible for a non-malicious user to lose money in our protocol because there is no TTP to resolve a dispute, and time-locks have their own drawbacks hence our protocol does not provide timeliness. Achieving two-party or multi-party fair exchange with *strong* fairness in the Ethereum ecosystem in the context of digital content exchange can be researched further.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[2] V. Buterin, “Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform,” *Ethereum*, 2013. .

[3] A. M. Antonopoulos, *The Internet of Money: A collection of talks by Andreas M. Antonopoulos*, 1st Editio. Merkle Bloom LLC, 2016.

[4] G. Chen, B. Xu, M. Lu, and N.-S. Chen, “Exploring blockchain technology and its potential applications for education,” *Smart Learn. Environ.*, vol. 5, no. 1, p. 1, Dec. 2018.

[5] K. Wust and A. Gervais, “Do you need a blockchain?,” in *Proceedings - 2018 Crypto Valley Conference on Blockchain Technology, CVCBT 2018*, 2018, pp. 45–54.

[6] J. D. Tygar and J. D., “Atomicity in electronic commerce,” in *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing - PODC ’96*, 1996, pp. 8–26.

[7] N. Asokan, “Fairness in electronic commerce,” University of Waterloo, 1998.

[8] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State, “Blockchain-Based, Decentralized Access Control for IPFS,” in *2018 IEEE International Conference on Blockchain (Blockchain 2018)*, 2018, pp. 1499–1506.

[9] G. Blank and B. C. Reisdorf, “The Participatory Web: A user perspective on Web 2.0,” *Information Communication and Society*, vol. 15, no. 4. pp. 537–554, May-2012.

[10] J. Isaak and M. J. Hanna, “User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection,” *Computer (Long. Beach. Calif).*, vol. 51, no. 8, pp. 56–59, Aug. 2018.

[11] S. Wheatley, T. Maillart, and D. Sornette, “The extreme risk of personal data breaches and the erosion of privacy,” *Eur. Phys. J. B*, vol. 89, no. 1, p. 7, Jan. 2016.

[12] M. Crosby, Nachiappan, P. Pattanayak, S. Verma, and V. Kalyanaraman, “BlockChain Technology: Beyond Bitcoin,” 2016.

[13] C. Davis, “Realizing Software Reliability in the Face of Infrastructure Instability,” *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 34–40, Sep. 2017.

[14] J. Bort, “AWS outage hurt internet retailers except Amazon - Business Insider,” *Business Insider*, 2017. [Online]. Available: https://www.businessinsider.com/aws-outage-hurt-internet-retailers-except-amazon-2017-3. [Accessed: 01-Jul-2019].

[15] Dyn, “Dyn Analysis Summary Of Friday October 21 Attack | Dyn Blog,” *Dyn Company News*, 2016. [Online]. Available: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/. [Accessed: 01-Jul-2019].

[16] Cloudflare, “Famous DDoS Attacks | Cloudflare,” 2018. [Online]. Available: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/. [Accessed: 05-Jul-2019].

[17] C.-W. Chu, “Censorship or Protectionism? Reassessing China’s Regulation of Internet Industry,” *Int. J. Soc. Sci. Humanit.*, vol. 7, no. 1, pp. 28–32, 2017.

[18] J. Zittrain, R. Faris, H. Noman, J. Clark, C. Tilton, and R. Morrison-Westphal, “The Shifting Landscape of Global Internet Censorship An Uptake in Communications Encryption Is Tempered by Increasing Pressure on Major Platform Providers; Governments Expand Content Restriction Tactics,” 2017.

[19] C. Doctorow, “EU Internet Censorship Will Censor the Whole World’s Internet | Electronic Frontier Foundation,” *Electronic Frontier Foundation*, 2018. [Online]. Available: https://www.eff.org/tr/deeplinks/2018/10/eu-internet-censorship-will-censor-whole-worlds-internet. [Accessed: 05-Jul-2019].

[20] G. Zyskind, O. Nathan, and A. “Sandy” Pentland, “Decentralizing privacy: Using blockchain to protect personal data,” in *Proceedings - 2015 IEEE Security and Privacy Workshops, SPW 2015*, 2015, pp. 180–184.

[21] H. Halpin and M. Piekarska, “Introduction to Security and Privacy on the Blockchain,” in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2017, pp. 1–3.

[22] V. Buterin, “Privacy on the Blockchain,” 2016. [Online]. Available: https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/. [Accessed: 28-Feb-2019].

[23] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, “Zcash Protocol Specification,” Tech. rep. 2016–1.10, 2016.

[24] R. Mercer, “Privacy on the Blockchain: Unique Ring Signatures,” Dec. 2016.

[25] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts,” in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 839–858.

[26] G. Zyskind, O. Nathan, and A. Pentland, “Enigma: Decentralized Computation Platform with Guaranteed Privacy,” 2015.

[27] V. Patel, “A framework for secure and decentralized sharing of medical imaging data via blockchain consensus,” *Health Informatics Journal*, SAGE PublicationsSage UK: London, England, p. 146045821876969, 25-Apr-2018.

[28] T. T. Thwin and S. Vasupongayya, “Blockchain Based Secret-Data Sharing Model for Personal Health Record System,” in *2018 5th International Conference on Advanced Informatics: Concept Theory and Applications (ICAICTA)*, 2018, pp. 196–201.

[29] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, “MedBlock: Efficient and Secure Medical Data Sharing Via Blockchain,” *J. Med. Syst.*, vol. 42, no. 8, p. 136, Aug. 2018.

[30] H. Yang and B. Yang, “A Blockchain-based Approach to the Secure Sharing of Healthcare Data,” *Proc. Nor. Inf. Secur.*, 2017.

[31] S. Wu and J. Du, “Electronic medical record security sharing model based on blockchain,” 2019, pp. 13–17.

[32] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, “FHIRChain: Applying Blockchain to Securely and Scalably Share Clinical Data,” *Comput. Struct. Biotechnol. J.*, vol. 16, pp. 267–278, Jan. 2018.

[33] S. Amofa *et al.*, “A blockchain-based architecture framework for secure sharing of personal health data,” in *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services, Healthcom 2018*, 2018, pp. 1–6.

[34] M. A. Cyran, “Blockchain as a Foundation for Sharing Healthcare Data,” *Blockchain Healthc. Today*, vol. 1, no. 0, Mar. 2018.

[35] S. Cui, M. R. Asghar, and G. Russello, “Towards blockchain-based scalable and trustworthy file sharing,” in *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 2018, vol. 2018–July.

[36] I. Barclay, “Innovative Applications of Blockchain Technology in Crime and Security,” Cardiff University, 2017.

[37] S. Wang, Y. Zhang, and Y. Zhang, “A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems,” *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[38] M. R. Hoffman, L. D. Ibáñez, H. Fryer, and E. Simperl, “Smart Papers: Dynamic Publications on the Blockchain,” in *The Semantic Web. ESWC 2018.*, 2018, vol. 10843 LNCS, pp. 304–318.

[39] G. Ateniese and Giuseppe, “Efficient verifiable encryption (and fair exchange) of digital signatures,” in *Proceedings of the 6th ACM conference on Computer and communications security - CCS ’99*, 1999, pp. 138–146.

[40] N. Francez, *Fairness*, Texts and. Springer, 1986.

[41] N. Asokan, M. Schunter, and M. Waidner, “Optimistic protocols for fair exchange,” in *Proceedings of the 4th ACM conference on Computer and communications security - CCS ’97*, 1997, pp. 7–17.

[42] H. Pagnia and F. C. Gärtner, “On the Impossibility of Fair Exchange without a Trusted Third Party,” Technical Report TUD-BS-1999-02, 1999.

[43] H. Bürk and A. Pfitzmann, “Value exchange systems enabling security and unobservability,” *Comput. Secur.*, vol. 9, no. 8, pp. 715–721, Dec. 1990.

[44] N. Asokan, V. Shoup, and M. Waidner, “Optimistic fair exchange of digital signatures,” in *Advances in Cryptology - EUROCRYPT’98,* 1998, vol. 1403, pp. 591–606.

[45] Feng Bao, R. H. Deng, and Wenbo Mao, “Efficient and practical fair exchange protocols with off-line TTP,” in *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*, 1998, pp. 77–85.

[46] A. Küpçü and A. Lysyanskaya, “Optimistic fair exchange with multiple arbiters,” in *Computer Security – ESORICS 2010.*, 2010, vol. 6345 LNCS, pp. 488–507.

[47] A. Küpçü and A. Lysyanskaya, “Usable optimistic fair exchange,” *Comput. Networks*, vol. 56, no. 1, pp. 50–63, Jan. 2012.

[48] A. Küpçü, “Distributing trusted third parties,” *ACM SIGACT News*, vol. 44, no. 2, pp. 92–112, Jun. 2013.

[49] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, “Zero-Knowledge Contingent Payments Revisited,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS ’17*, 2017, pp. 229–243.

[50] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, “Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin,” in *Financial Cryptography and Data Security. FC 2017. Lecture Notes in Computer Science*, 2017, vol. 10322 LNCS, pp. 321–339.

[51] S. Dziembowski, L. Eckey, and S. Faust, “Fairswap: How to fairly exchange digital goods,” in *Proceedings of the ACM Conference on Computer and Communications Security*, 2018, pp. 967–984.

[52] I. Bentov and R. Kumaresan, “How to use Bitcoin to design fair protocols,” in *Advances in Cryptology – CRYPTO 2014*, 2014, vol. 8617 LNCS, no. PART 2, pp. 421–439.

[53] E. Wagner, A. Volker, F. Fuhrmann, R. Matzutt, and K. Wehrle, “Dispute Resolution for Smart Contract-based Two-Party Protocols,” in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 422–430.

[54] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek, “Fair two-party computations via bitcoin deposits,” in *Financial Cryptography and Data Security. FC 2014. Lecture Notes in Computer Science*, 2014, vol. 8438, pp. 105–121.

[55] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, “Secure Multiparty Computations on Bitcoin,” in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 443–458.

[56] S. Haber and W. S. Stornetta, “How to time-stamp a digital document,” *J.*

Cryptol., vol. 3, no. 2, pp. 99–111, 1991.

[57] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the Efficiency and Reliability of Digital Time-Stamping," 1992.

[58] Z. Zheng, S. Xie, H. N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: a survey," *Int. J. Web Grid Serv.*, vol. 14, no. 4, p. 352, 2018.

[59] V. Buterin, "Merkling in Ethereum," *Ethereum Blog*, 2015. [Online]. Available: https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/. [Accessed: 29-Jul-2019].

[60] Zcash, "What are zk-SNARKs? | Zcash," 2019. [Online]. Available: https://z.cash/technology/zksnarks/. [Accessed: 31-May-2019].

[61] Monero, "What is Monero (XMR)? | Monero - secure, private, untraceable," 2018. [Online]. Available: https://web.getmonero.org/get-started/what-is-monero/. [Accessed: 31-May-2019].

[62] M. Swan, *Blockchain : blueprint for a new economy*. O'Reilly Media, 2015.

[63] M. Russinovich, "Digitizing trust: Azure Blockchain Service simplifies blockchain development," *Microsoft Azure Blog*, 2019. [Online]. Available: https://azure.microsoft.com/en-us/blog/digitizing-trust-azure-blockchain-service-simplifies-blockchain-development/. [Accessed: 05-Jul-2019].

[64] AWS, "Blockchain on AWS," *Amazon Web Services*, 2019. [Online]. Available: https://aws.amazon.com/blockchain/. [Accessed: 05-Jul-2019].

[65] Google, "Ethereum (Google Click to Deploy)," *Google Cloud Platform*, 2019. [Online]. Available: https://console.cloud.google.com/marketplace/details/click-to-deploy-images/ethereum. [Accessed: 05-Jul-2019].

[66] A. Baliga, "Understanding Blockchain Consensus Models," *Whitepaper*, pp. 1–14, 2017.

[67] W. Wang *et al.*, "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks," *IEEE Access*, vol. 7, pp. 22328–22370, 2019.

[68] S. Chu and S. Wang, "The Curses of Blockchain Decentralization," Oct. 2018.

[69] Y. Kwon, J. Liu, M. Kim, D. Song, and Y. Kim, "Impossibility of Full Decentralization in Permissionless Blockchains," May 2019.

[70] A. Gervais, G. O. Karame, V. Capkun, and S. Capkun, "Is Bitcoin a Decentralized Currency?," *IEEE Secur. Priv.*, vol. 12, no. 3, pp. 54–60, May 2014.

[71] Don Tapscott and Alex Tapscott, "The Impact of the Blockchain Goes Beyond Financial Services," *Harv. Bus. Rev.*, p. 7, 2016.

[72] Binance, "Transaction Information," 2019. [Online]. Available: https://explorer.binance.org/tx/00B41E4CCB8B977C78E20C19B8A0E39E57

2FCC80F53AA009163FA08B0AD57DE9. [Accessed: 26-Jun-2019].

[73] D. Seaman, *The Bitcoin Primer: Risks, Opportunities, And Possibilities*. Amazon Digital Services LLC., 2013.

[74] S. Ølnes, J. Ubacht, and M. Janssen, "Blockchain in government: Benefits and implications of distributed ledger technology for information sharing," *Government Information Quarterly*, vol. 34, no. 3, JAI, pp. 355–364, 01-Sep-2017.

[75] Bitinfocharts, "Bitcoin (BTC) statistics - Price, Blocks Count, Difficulty, Hashrate, Value," 2019. [Online]. Available: https://bitinfocharts.com/bitcoin/. [Accessed: 28-May-2019].

[76] Etherscan, "Ethereum ChainData Size Growth - Fast Sync," 2019. [Online]. Available: https://etherscan.io/chart2/chaindatasizefast. [Accessed: 28-May-2019].

[77] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," 2019.

[78] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer, "Decentralization in Bitcoin and Ethereum Networks," Jan. 2018.

[79] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, 2017.

[80] P. Otte, M. de Vos, and J. Pouwelse, "TrustChain: A Sybil-resistant scalable blockchain," *Future Generation Computer Systems*, Sep-2017.

[81] I. C. Lin and T. C. Liao, "A survey of blockchain security issues and challenges," *Int. J. Netw. Secur.*, vol. 19, no. 5, pp. 653–659, 2017.

[82] F. Gai, B. Wang, W. Deng, and W. Peng, "Proof of reputation: A reputation-based consensus protocol for peer-to-peer network," in *Database Systems for Advanced Applications. DASFAA 2018.*, 2018, vol. 10828 LNCS, pp. 666–681.

[83] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, pp. 557–564.

[84] A. M. Antonopoulos and G. Wood, *Mastering Ethereum*, 1st Editio. O'Reilly Media, 2018.

[85] N. Atzei, M. Bartoletti, and T. Cimoli, "A Survey of Attacks on Ethereum Smart Contracts (SoK) BT - Principles of Security and Trust," in *International Conference on Principles of Security and Trust*, 2017, pp. 164–186.

[86] Ethereum Community, "Ethereum Homestead Documentation Release 0.1," 2017.

[87] Truffle, "Interacting with Your Contracts," *Truffle Documentation*, 2019. [Online]. Available: https://www.trufflesuite.com/docs/truffle/getting-started/interacting-with-your-contracts. [Accessed: 26-Jul-2019].

[88] N. Atzei, M. Bartoletti, T. Cimoli, S. Lande, and R. Zunino, “SoK: Unraveling bitcoin smart contracts,” in *Principles of Security and Trust. POST 2018.*, 2018, vol. 10804 LNCS, pp. 217–242.

[89] district0x, “The Role of Tokens in Ethereum | Understanding Ethereum Tokens,” 2019. [Online]. Available: https://education.district0x.io/general-topics/understanding-ethereum/the-role-of-tokens/. [Accessed: 28-Jun-2019].

[90] Etherscan, “Etherscan Token Tracker,” 2019. [Online]. Available: https://etherscan.io/tokens. [Accessed: 28-Jun-2019].

[91] Icotoplist.com, “ICO List - Top List of Best ICOs in 2019,” 2019. [Online]. Available: https://icotoplist.com/. [Accessed: 28-Jun-2019].

[92] G. Fenu, L. Marchesi, M. Marchesi, and R. Tonelli, “The ICO phenomenon and its relationships with ethereum smart contract environment,” in *2018 IEEE 1st International Workshop on Blockchain Oriented Software Engineering, IWBOSE 2018 - Proceedings*, 2018, vol. 2018–Janua, pp. 1–7.

[93] S. Bian *et al.*, “IcoRating: A Deep-Learning System for Scam ICO Identification,” 2018.

[94] Solidity, “Solidity Documentation,” 2019. [Online]. Available: https://solidity.readthedocs.io/en/latest/index.html. [Accessed: 08-Jun-2019].

[95] V. Buterin, “Vyper Documentation,” 2018.

[96] I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, “Finding The Greedy, Prodigal, and Suicidal Contracts at Scale,” 2018.

[97] F. Klint, “A $50 Million Hack Just Showed That the DAO Was All Too Human | WIRED,” *WIRED.com*, 2016. [Online]. Available: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/. [Accessed: 09-Jun-2019].

[98] DASP, “DASP - TOP 10,” 2018. [Online]. Available: https://dasp.co/. [Accessed: 09-Jun-2019].

[99] OpenZeppelin, “OpenZeppelin,” 2019. [Online]. Available: https://openzeppelin.org/. [Accessed: 09-Jun-2019].

[100] ConsenSys Diligence, “Ethereum Smart Contract Best Practices,” 2019. [Online]. Available: https://consensys.github.io/smart-contract-best-practices/. [Accessed: 09-Jun-2019].

[101] Solidity, “Security Considerations — Solidity 0.5.10 documentation,” 2019. [Online]. Available: https://solidity.readthedocs.io/en/latest/security-considerations.html#. [Accessed: 09-Jun-2019].

[102] J. Stark, E. Van Ness, and D. Zakrisson, “The Year in Ethereum,” 2019. [Online]. Available: https://medium.com/@jjmstark/the-year-in-ethereum-87a17d6f8276. [Accessed: 09-Jun-2019].

[103] M. Ober, S. Katzenbeisser, and K. Hamacher, “Structure and Anonymity of the

Bitcoin Transaction Graph," *Futur. Internet*, vol. 5, pp. 237–250, 2013.

[104] N. Kshetri, "Blockchain's roles in strengthening cybersecurity and protecting privacy," *Telecomm. Policy*, vol. 41, no. 10, pp. 1027–1038, Nov. 2017.

[105] S. Goldfeder, H. Kalodner, D. Reisman, and A. Narayanan, "When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies," *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 4, pp. 179–199, Oct. 2018.

[106] Smolenski Michael, "Lightstreams White Paper," 2018.

[107] S. Tual, "Web 3.0 Revisited — Part One: 'Across Chains and Across Protocols,'" 2017. [Online]. Available: https://blog.stephantual.com/web-3-0-revisited-part-one-across-chains-and-across-protocols-4282b01054c5. [Accessed: 31-May-2019].

[108] StateoftheDApps, "State of the DApps," 2019. [Online]. Available: https://www.stateofthedapps.com/. [Accessed: 03-Jun-2019].

[109] web3.js, "Web3.js - Ethereum JavaScript API," *web3.js Documentation*, 2018. [Online]. Available: https://web3js.readthedocs.io/en/1.0/. [Accessed: 31-May-2019].

[110] Metamask.io, "MetaMask," 2019. [Online]. Available: https://metamask.io/. [Accessed: 01-Jun-2019].

[111] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," 2014.

[112] B. Skvorc *et al.*, *Learn Ethereum : The Collection*, 1st ed. SitePoint, 2018.

[113] IPFS, "IPFS is the Distributed Web," 2019. [Online]. Available: ipfs.io. [Accessed: 31-May-2019].

[114] V. Tron, A. Fischer, N. Johnson, D. A. Nagy, and Z. Felfo, "Swarm Documentation Release 0.3," 2019.

[115] Swarm, "Swarm Website," 2019. [Online]. Available: https://swarm-gateways.net/bzz:/theswarm.eth/. [Accessed: 04-Jun-2019].

[116] ENS, "Introduction - Ethereum Name Service," 2019. [Online]. Available: https://docs.ens.domains/. [Accessed: 04-Jun-2019].

[117] Whisper, "Whisper PoC 2 Protocol Spec," 2018. [Online]. Available: https://github.com/ethereum/wiki/wiki/Whisper-PoC-2-Protocol-Spec. [Accessed: 03-Jun-2019].

[118] Oraclize, "Oraclize Documentation," 2019. [Online]. Available: https://docs.oraclize.it/. [Accessed: 05-Jun-2019].

[119] S. Ellis, A. Juels, and S. Nazarov, "ChainLink: A Decentralized Oracle Network," vol. 2017, no. September, pp. 1–38, 2017.

[120] M. Alharby and A. van Moorsel, "Blockchain Based Smart Contracts : A Systematic Mapping Study," in *Computer Science & Information Technology (CS & IT)*, 2017, pp. 125–140.

[121] T. T. Kuo, H. Zavaleta Rojas, and L. Ohno-Machado, “Comparison of blockchain platforms: A systematic review and healthcare examples,” *Journal of the American Medical Informatics Association*, vol. 26, no. 5. pp. 462–478, 2019.

[122] M. Macdonald, L. Liu-Thorrold, and R. Julien, “The Blockchain: A Comparison of Platforms and Their Uses Beyond Bitcoin,” no. February, pp. 1–18, 2017.

[123] GnuPG, “The GNU Privacy Guard,” 2019. [Online]. Available: https://gnupg.org/. [Accessed: 09-Sep-2019].

[124] D. Huigens, “Cure53 security audit,” *OpenPGP.js Wiki*, 2018. [Online]. Available: https://github.com/openpgpjs/openpgpjs/wiki/Cure53-security-audit. [Accessed: 27-Jul-2019].

[125] G. Asharov, Y. Lindell, and H. Zarosim, “Fair and efficient secure multiparty computation with reputation systems,” in *Advances in Cryptology - ASIACRYPT (2)*, 2013, pp. 201–220.

[126] E. J. Friedman and P. Resnick, “The Social Cost of Cheap Pseudonyms,” *J. Econ. Manag. Strateg.*, vol. 10, no. 2, pp. 173–199, Jun. 2001.

[127] X. Hu, Z. Lin, A. B. Whinston, and H. Zhang, “Hope or Hype: On the Viability of Escrow Services as Trusted Third Parties in Online Auction Environments,” *Inf. Syst. Res.*, vol. 15, no. 3, pp. 236–249, 2004.

[128] D. Zimbeck, “Two Party double deposit trustless escrow in cryptographic networks and Bitcoin,” 2014.

[129] A. Asgaonkar and B. Krishnamachari, “Solving the Buyer and Seller’s Dilemma: A Dual-Deposit Escrow Smart Contract for Provably Cheat-Proof Delivery and Payment for a Digital Good without a Trusted Mediator,” in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 262–267.

[130] Solidity Documentation, “Safe Remote Purchase,” 2017. [Online]. Available: https://solidity.readthedocs.io/en/v0.4.18/solidity-by-example.html#safe-remote-purchase. [Accessed: 29-Jul-2019].

[131] S. J. Ong, D. C. Parkes, A. Rosen, and S. Vadhan, “Fairness with an honest minority and a rational majority,” in *Theory of Cryptography. TCC 2009.*, 2009, vol. 5444 LNCS, pp. 36–53.

[132] J. Halpern and V. Teague, “Rational secret sharing and multiparty computation,” in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing - STOC ’04*, 2004, p. 623.

[133] Solidity, “Safe Remote Purchase,” *Solidity Documentation*, 2019. [Online]. Available: https://solidity.readthedocs.io/en/latest/solidity-by-example.html#safe-remote-purchase. [Accessed: 03-Aug-2019].

[134] OpenPGP.js, “Encrypt and decrypt String data with PGP keys,” *OpenPGP.js Documentation*, 2019. [Online]. Available:

https://github.com/openpgpjs/openpgpjs/blob/master/README.md#encrypt-and-decrypt-string-data-with-pgp-keys. [Accessed: 03-Aug-2019].

[135] E. Barker, “Recommendation for Key Management Part 1: General - NIST Special Publication 800-57 Part 1 Revision 4,” 2016.

[136] ETH Gas Station, “Consumer oriented metrics for the Ethereum gas market,” *ethgasstation.info*, 2019. [Online]. Available: https://ethgasstation.info/. [Accessed: 06-Aug-2019].

[137] V. Buterin, “Sharding FAQ,” *Ethereum Wiki*, 2019. [Online]. Available: https://github.com/ethereum/wiki/wiki/Sharding-FAQ. [Accessed: 21-Aug-2019].

[138] V. Buterin, “On Slow and Fast Block Times,” *ethereum.org*, 2015. [Online]. Available: https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/. [Accessed: 09-Aug-2019].

APPENDIX

SOURCE CODE OF MANAGEMENT AND PURCHASE SMART CONTRACTS

```
// Our ContractManagement contract is a typical “contract factory” implementation that spawns new contracts and holds their addresses in an array.

// Our Purchase contract source code is based on the “Safe Remote Purchase” example in official Ethereum documentation.
// url: https://solidity.readthedocs.io/en/v0.5.11/solidity-by-example.html#safe-remote-purchase
// August 2019


pragma solidity ^0.5.11;

// this contract acts as a contract factory to create new Purchase contracts
contract ContractManagement {
    address[] public deployedContracts;
    address public owner;

    // seller creates a Purchase escrow contract to put a content on sale.
    // Put content on sale use case.
    function newPurchaseContract(
        string memory _name,
        string memory _description,
        string memory _contentHashAddress)
    public
    payable
    returns(address _newContract)
    {
        Purchase newContract = (new Purchase).value(msg.value)(
            msg.sender,
            _name,
            _description,
            _contentHashAddress);
        deployedContracts.push(address(newContract));
        return address(newContract);
```

```
    }

    // get the total number of escrow contracts to use in a loop in the frontend
    function getPurchaseCount()
    public
    view
    returns(uint _contractCount)
    {
        return deployedContracts.length;
    }

    // return the address of a particular escrow contract according to its index,
    // so that users can list and interact with the contracts from the frontend
    function getPurchaseAddress(uint i)
    public
    view
    returns(address _contractAddress)
    {
        return deployedContracts[i];
    }
}


// this contract acts as the double escrow trustless mediator to ensure fair exchange
contract Purchase {
    // state variables
    uint public value;
    address payable public seller;
    address payable public buyer;
    string public name;
    string public description;
    string public secretHashAddress;
    string public contentHashAddress;
    string public buyerPublicKey;
    enum State { Created, Locked, Inactive }
    // The state variable has a default value of the first member, `State.created`
    State public state;


    // Put content on sale use case.
    // Ensure that `msg.value` which is the deposit is an even number.
    constructor(
        address payable _seller,
        string memory _name,
        string memory _description,
        string memory _contentHashAddress)
    public
    payable
```

```
    {
        seller = _seller;
        name = _name;
        description = _description;
        contentHashAddress = _contentHashAddress;

        value = msg.value / 2;
        require((2 * value) == msg.value, "Deposit amount has to be twice of the
price.");

        emit LogSellContent(seller, name, value);
    }

    // Modifiers for user authorization
    modifier condition(bool _condition) {
        require(_condition);
        _;
    }

    modifier onlyBuyer() {
        require(
            msg.sender == buyer,
            "Only buyer can call this."
        );
        _;
    }

    modifier onlySeller() {
        require(
            msg.sender == seller,
            "Only seller can call this."
        );
        _;
    }

    modifier inState(State _state) {
        require(
            state == _state,
            "Invalid state."
        );
        _;
    }

    // emit events for information purposes required by the frontend
    event LogSellContent(
        address indexed _seller,
        string _name,
        uint _value
```

```
);

event LogBuyContent(
    address indexed _buyer,
    string _name,
    uint _value
);

event LogUploadSecret(
    address indexed _seller,
    string _name,
    string _secretHashAddress
);

event LogContentReceived(
    address indexed _buyer,
    string _name
);

event LogAborted(
    address indexed _seller,
    string _name
);


// Abort the purchase and reclaim the deposit.
function abort()
public
onlySeller
inState(State.Created)
{
    emit LogAborted(seller, name);
    state = State.Inactive;
    seller.transfer(address(this).balance);
}


// Buyer confirms the purchase and buys the content
function buyContent(string memory _buyerPublicKey)
public
inState(State.Created)
condition(msg.sender != seller)
condition(msg.value == (2 * value))
payable
{
    emit LogBuyContent(buyer, name, value);
    buyer = msg.sender;
    state = State.Locked;
```

```
        buyerPublicKey = _buyerPublicKey;
    }

    // seller uploads the secret for content delivery use case.
    function uploadSecret(string memory _secretHashAddress)
    public
    onlySeller
    inState(State.Locked)
    {
        emit LogUploadSecret(seller, name, secretHashAddress);
        secretHashAddress = _secretHashAddress;
    }


    // Buyer Confirms Received (release deposits)
    function confirmReceived()
    public
    onlyBuyer
    inState(State.Locked)
    {
        emit LogContentReceived(buyer, name);
        // change state first for security
        state = State.Inactive;

        buyer.transfer(value);
        seller.transfer(address(this).balance);
    }


    // get the details of the content to display in the frontend
    function getContent()
    public
    view
    returns(
        address _seller,
        address _buyer,
        string memory _name,
        string memory _description,
        string memory _secretHashAddress,
        uint _value,
        uint8 _state,
        string memory _contentHashAddress,
        string memory _buyerPublicKey)
    {
        return(seller, buyer, name, description, secretHashAddress, value, uint8(state),
contentHashAddress, buyerPublicKey);
    }
}
```

TEZ İZİN FORMU / THESIS PERMISSION FORM

ENSTİTÜ / INSTITUTE

Fen Bilimleri Enstitüsü / Graduate School of Natural and Applied Sciences ☐

Sosyal Bilimler Enstitüsü / Graduate School of Social Sciences ☐

Uygulamalı Matematik Enstitüsü / Graduate School of Applied Mathematics ☐

Enformatik Enstitüsü / Graduate School of Informatics ☐

Deniz Bilimleri Enstitüsü / Graduate School of Marine Sciences ☐

YAZARIN / AUTHOR

Soyadı / Surname : .....
Adı / Name : .....
Bölümü / Department : .....

TEZİN ADI / TITLE OF THE THESIS (**İngilizce** / English) : .....
.....
.....
.....
.....

TEZİN TÜRÜ / DEGREE**:** **Yüksek Lisans** / Master ☐ **Doktora** / PhD ☐

1. **Tezin tamamı dünya çapında erişime açılacaktır. /** Release the entire work immediately for access worldwide. ☐
2. **Tez iki yıl süreyle erişime kapalı olacaktır.** / Secure the entire work for patent and/or proprietary purposes for a period of **two year. *** ☐
3. **Tez altı ay süreyle erişime kapalı olacaktır.** / Secure the entire work for period of **six months**. ***** ☐

***** *Enstitü Yönetim Kurulu Kararının basılı kopyası tezle birlikte kütüphaneye teslim edilecektir.*
A copy of the Decision of the Institute Administrative Committee will be delivered to the library together with the printed thesis.

Yazarın imzası / Signature ..... **Tarih** / Date .....