UNIVERSITY OF MACEDONIA
GRADUATE PROGRAM
DEPARTMENT OF APPLIED INFORMATICS

DECENTRALIZED APPLICATIONS: DEVELOPMENT OF A BLOCKCHAIN-BASED ACADEMIC DOCUMENTS VERIFICATION PLATFORM

A master thesis presented by

Argyrios Margaritis

Thessaloniki, August 2021

DECENTRALIZED APPLICATIONS: DEVELOPMENT OF A BLOCKCHAIN-BASED ACADEMIC DOCUMENTS VERIFICATION PLATFORM

Argyrios Margaritis

Degree in Applied Informatics, University of Macedonia, 2018

Master Thesis

submitted for the partial fulfillment of the

DEGREE OF MASTER OF SCIENCE IN APPLIED INFORMATICS

Supervisor
Professor Alexander Chatzigeorgiou

Approved by examining board on November 1, 2021

Prof. Alexander Chatzigeorgiou	Prof. Emmanouil Stiakakis	Prof. Ioannis Mavridis
.....	.....	.....

Argyrios Margaritis

AM

.....

Abstract

Decentralized applications comprise a significant next step towards the new web. Web 3.0 is a leap forward to open, trustless and permissionless networks. This dissertation covers how cryptography can lay the foundation for the decentralized web to become a reality. We follow the contribution of groundbreaking technologies like the blockchain to a new domain of software engineering. Smart contracts development emerges through second-generation blockchain platforms like Ethereum. Today, there is a broad ecosystem of tools participating in dapp development. Complementary decentralized technologies like the IPFS can assist in the implementation of feature-complete solutions.

Furthermore, this thesis showcases the development of a decentralized academic certificates registry prototype by the name CryptoCerts. CryptoCerts allows the transparent and verifiable distribution of academic documents in an immutable and trustless manner. The system consists of a modern web client written in React and smart contracts written in Solidity deployed on the Ethereum blockchain. CryptoCerts demonstrates how IPFS can be utilized for decentralized documents storage and how users can navigate a dapp with an inclusive user experience. Last, we review the challenges of coding cost-efficient smart contracts and suggest topics for future work.

Keywords: decentralized application, dapp, smart contracts, blockchain, Ethereum, Solidity, IPFS, documents, certificates, verification, software engineering, development, React, MetaMask

Περίληψη

Οι αποκεντρωμένες εφαρμογές αποτελούν ένα σημαντικό νέο βήμα προς μια νέα μορφή του διαδικτύου. Το Web 3.0 αποτελεί ένα βήμα προς δίκτυα ανοιχτά, μη βασιζόμενα στην εμπιστοσύνη προς τρίτους και χωρίς την ανάγκη αδειοδότησης. Αυτή η εργασία καλύπτει το πως η κρυπτογραφία μπορεί να θέσει τα θεμέλια για να γίνει το αποκεντρωμένο διαδίκτυο πραγματικότητα. Ακολουθούμε τη συνεισφορά πρωτοποριακών τεχνολογιών όπως το blockchain στη δημιουργία ενός νέου τομέα της τεχνολογίας λογισμικού. Η ανάπτυξη έξυπνων συμβολαίων αναδύεται μέσα από τις πλατφόρμες blockchain δεύτερης γενιάς όπως το Ethereum. Σήμερα, ένα ευρύ οικοσύστημα από εργαλεία ανάπτυξης συνεισφέρουν στην ανάπτυξη αποκεντρωμένων εφαρμογών. Άλλες εξίσου αποκεντρωμένες τεχνολογίες όπως το IPFS μπορούν να συνδράμουν συμπληρωματικά στην υλοποίηση ολοκληρωμένων λύσεων.

Επιπρόσθετα, αυτή η διπλωματική εργασία παρουσιάζει την ανάπτυξη ενός πρότυπου αποκεντρωμένου μητρώου καταχώρησης ακαδημαϊκών πιστοποιητικών με το όνομα CryptoCerts. Το CryptoCerts επιτρέπει τη διαφανή και επαληθεύσιμη διανομή ακαδημαϊκών εγγράφων με τρόπο αδιάβλητο και χωρίς την ανάγκη εμπιστοσύνης σε τρίτα μέρη. Το σύστημα αποτελείται από έναν σύγχρονο πρόγραμμα πελάτη με τεχνολογίες ιστού γραμμένο σε React καθώς και από έξυπνα συμβόλαια γραμμένα σε Solidity και εγκατεστημένα στη blockchain πλατφόρμα του Ethereum. Το CryptoCerts αναδεικνύει πως μπορεί να χρησιμοποιηθεί το IPFS σαν μία αποκεντρωμένη λύση αποθήκευσης εγγράφων καθώς και πως οι χρήστες μπορούν να χρησιμοποιήσουν μια αποκεντρωμένη εφαρμογή με μια εμπειρία χρήστη χωρίς αποκλεισμούς. Τέλος, αναλύουμε τις προκλήσεις του προγραμματισμού οικονομικά αποδοτικών έξυπνων συμβολαίων και προτείνουμε θέματα για μελλοντικές επεκτάσεις.

Λέξεις Κλειδιά: αποκεντρωμένες εφαρμογές, dapps, έξυπνα συμβόλαια, blockchain, Ethereum, Solidity, IPFS, έγγραφα, πιστοποιητικά, επαλήθευση, τεχνολογία λογισμικού, React, MetaMask

Preface – Acknowledgements

Before you lies the dissertation "Decentralized Applications: Development of a Blockchain-based Academic Documents Verification Platform." It was originated from my long-time pursuit to enter into decentralized application development and acquire hands-on experience with cutting-edge developments in software engineering. It has been written to fulfil the graduation requirements of the Degree of Master of Science in Applied Informatics. I was engaged in researching and implementing, and writing this dissertation from July 2020 to August 2021.

I want to thank my supervisor, Prof. Alexander Chatzigeorgiou, for giving me the opportunity to cooperate on a topic so intriguing and innovative. His invaluable guidance and support served me well during this long process. Additionally, I would like to thank Prof. Ioannis Mavridis and Prof. Emmanouil Stiakakis for their unreserved participation in the examination board of this thesis. Last, I wish to thank my friends and family for supporting me during the hard times. If I ever lost interest, you kept me motivated.

I hope you enjoy your reading.

Argyris Margaritis

Thessaloniki, October 19, 2021

Table of Contents

Table of Figures

Table of Screenshots

Table of Code

Table of Wireframes

List of Abbreviations

IoT	Internet of Things
AWS	Amazon Web Services
SSO	Single Sign-On
NSA	National Security Agency
Dapp	Decentralized Application
P2P	Peer-To-Peer
SKC	Secret Key Cryptography
PKC	Public Key Cryptography
DES	Data Encryption Standard
IBM	International Business Machines
NBS	National Bureau of Standards
AES	Advanced Encryption Standard
RSA	Rivest, Shamir, & Adleman
DSA	Digital Signature Algorithm
ECC	Elliptical Curve Cryptography
MD	Message Digest
SHA	Secure Hash Algorithm
NIST	National Institute of Standards and Technology
CRUD	Create Update Delete
PoW	Proof of Work
PoS	Proof of Stake
PoA	Proof of Authority
DPoS	Delegated Proof of Stake
API	Application Programming Interface
KYC	Know Your Customer
EVM	Ethereum Virtual Machine
EOA	Externally Owned Account
ERC	Ethereum Request for Comment
NFT	Non-Fungible Tokens
SPDX	Software Package Data Exchange
JSON	JavaScript Object Notation

RPC	Remote Procedure Call
CLI	Command Line Interface
UI	User Interface
IPFS	InterPlanetary File System
DHT	Distributed Hash Tables
SFS	Self-certifying File System
IPLD	InterPlanetary Linked Data
DAG	Directed Acyclic Graph
CID	Content IDentifier
JSX	JavaScript XML
XML	eXtensible Markup Language
DOM	Document Object Model
VM	Virtual Machine
CPU	Central Processing Unit
YAML	Yet Another Markup Language
ACEI	Academic Credentials Evaluation Institute
MVP	Minimum Valuable Produce
PDF	Portable Document Format
UX	User eXperience
BIP	Bitcoin Improvement Proposal
ETH	ETHereum ticker
UML	Unified Modeling Language
CRA	Create React App
ES	ECMAScript
ECMA	European Computer Manufacturers Association
ABI	Application Binary Interface
zk-SNARK	Zero-Knowledge Succinct Non-interactive Argument of Knowledge

1 Introduction

1.1 The evolution of the web

About 30 years ago, following the extended evolution of computer networking, a new technological notion started spreading to a mainstream extend. A technology that was destined to shape the identity of the modern world. By the early 1990s, the World Wide Web will have emerged, connecting millions of computers around the globe.

The network of networks would be relatively static during its first years. Plain pages consisting solely of text and images would pop out of the computer screens presenting their host's perspective to the world. The first era of the web, commonly referred to as Web 1.0, offered no interactivity to its users. It shared many similarities with the traditional media like radio and television broadcasts and consisted of providers feeding content to a handful of passive surfers. Most websites were built using static HTML pages stored in files and a few embedded styles. A typical example of that design is depicted in [Figure 1].

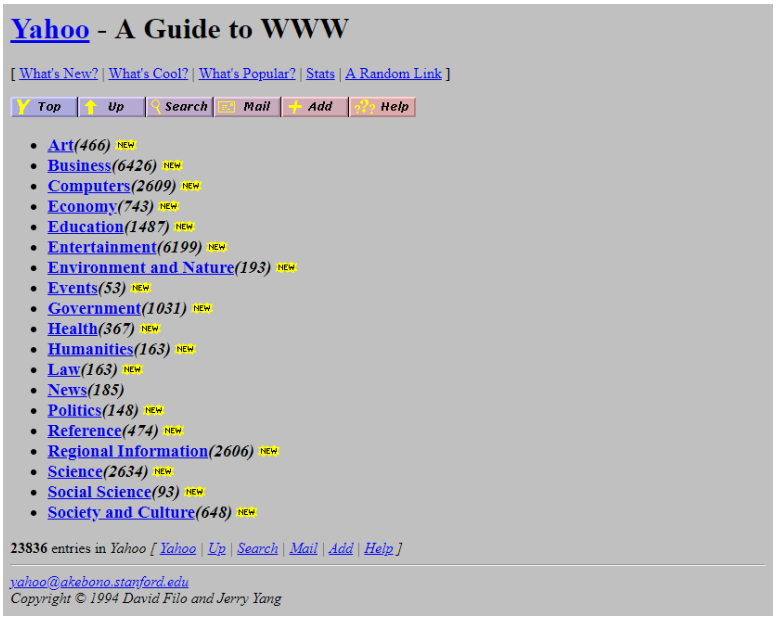


Figure 1 Yahoo in 1994 [69]

As the audiences grew more extensive than before, many opportunities emerged, and the future seemed without boundaries. Every business sector, organization, and community sought online representation. Mass excitement followed by fear of missing out led many to defy business fundamentals and fueled a frenzy that was about to burst in

early 2000. As it is widely known, the dot-com bubble, a severe financial crash came true with the technology-dominated NASDAQ index plummeting up to 77% in 2 years [1].

The crisis caused billions to vanish into thin air but let the web progress into a steadier boom. The web technology stack evolved, upgraded servers, and faster internet connections spread the expansion. Many websites implemented features to allow their users to contribute and generate content. Browsing got easier and data transmission more viable. The interactive web, the second stage in the internet evolution, came into existence. Web 2.0, commonly referred to nowadays, was coined during a brainstorming conference between O'Reilly and MediaLive International in 2005 [2].

It is necessary to clarify that Web 2.0 has no distinct boundaries but a core of principles and practices. During this stage, significant swifts happened on how software tends to be released, incorporating more agile and ongoing methodologies. In addition, more development companies moved away from the traditional software products to a more Software as a Service (SaaS) approach incorporating the value of data. Consequently, hosts are now the facilitators of user activities on their websites rather than the sole providers of static content. Notable examples of this are the social media networks and the media sharing platforms, which consist of the proven dominant portals of the new web.

The evolution of web technologies, along with the 3G high-speed mobile networks of the late 2000s, led to the rise of the mobile web. New portable, lightweight, yet powerful devices, smartphones, and wearables hit the mass market, accessing the internet from almost anywhere. Based on the annual report of the ITU for 2014 [3], the number of mobile-broadband users reached 2.3 billion, while more than half of them originated from developing countries.

The massive generation of unstructured data needed to be permanently stored and efficiently retrieved to be analyzed forced new types of storage to emerge and pushed the NoSQL databases into the foreground. Defined as Big Data, the new web era introduced a mixed concept for data preserved in significant volumes. As a result, the necessity of a new discipline dedicated to data cleansing, preparation, and analysis was revealed. The field of Data Science materialized [4].

1.2 The web today

As the dawn of the third decade in the 21st century arises, everyday life gets overrun by the astonishing achievements of the digital era. From work and finance to social relationships and entertainment, every aspect is affected by the impact of data permeating our lives through the internet. In the new global economy, digital growth is a significant indicator.

As illustrated in [Figure 2], there has been a steady rise in the number of users worldwide navigating the web over the past 15 years. In addition, these figures are expected to skyrocket during the current year, as the COVID-19 pandemic forced even more people to stay indoors and, by extension, online. There are reports [5] revealing a marked increase of 7.4% in the annual digital growth up to as of October 2020, approaching the significant percentage of 60% of the total world population.

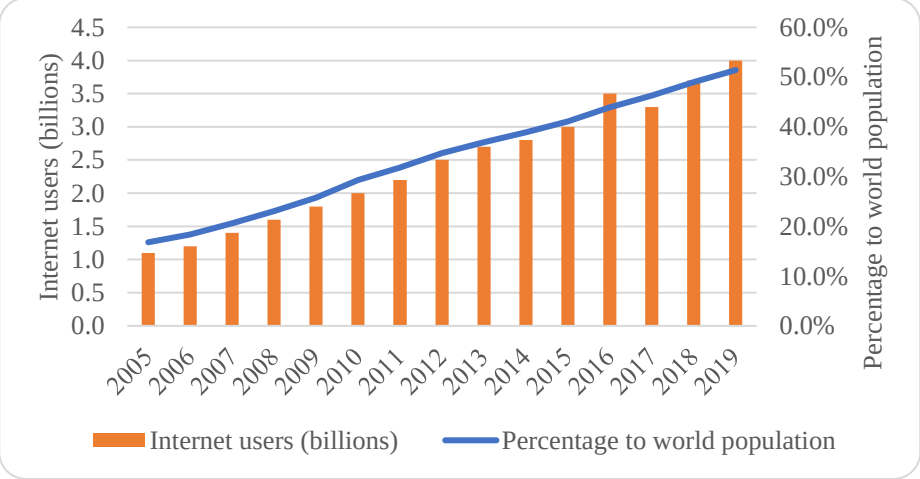


Figure 2 Internet usage since 2005. Adapted from [68]

As described in the previous section, the dot-com bubble burst in the early 2000s. A critical reader would have expected the financial crisis to have fragmented the technological businesses ever since. In contrast, today's web is dominated by a handful of tech giants worth incredible amounts of dollars. As an indicative example, Apple Inc. was the first one trillion dollars company in history by August 2018 [6]. The FAAMG (originated from the term FANG coined by Tim Cramer in 2013 [7]) is an acronym that stands for the five giant companies in the high-tech industry, namely Facebook, Amazon, Apple, Microsoft, and Google-parent Alphabet. As of July 2020, the FAAMG stocks

ruled the benchmark S&P 500 index by more than 22% to a greater extent than they did in 2000 [8].

Based on the numbers presented, it is clear that the digital world holds the most significant role in modern civilization. It is now necessary to step back and reflect on how all these are supported.

1.3 Centralization

From the earlier days of the web, the client-server model laid the foundation of the internet's interconnectivity. The client is a computer that sends a request, while the server is also a computer that responds to it by sending information back to the client. An important aspect that needs to be highlighted is the central position of the server in the client-server model. The server stands in the middle, serving requests simultaneously from the clients surrounding it. There is no way in this model for the clients to communicate with each other directly, even if they are located next to each other in the physical world.

Today's clients consist of billions of users surfing the web from their computers or smartphones and the massive number of IoT devices from sensors to smart cars or refrigerators. Similarly, the servers have also evolved. From dedicated computers run by individuals to host their website during Web 1.0, the vast need for performance to efficiently respond to the unimaginable number of requests moved the servers from houses to gigantic data centers. The picture drawn is one of the billions of clients served every second by massive racks of servers co-located in specific places. Predictably, this creates a centralized model with servers playing a critical role and is often described as the single point of failure.

1.4 The need for a decentralized web

The perils of centralization have been seen in the cases of significant incidents experienced on cloud providers comprising the backbone of the internet. On November 25^{th,} 2020, a flaw in the Kinesis Data Streams API of the Amazon Web Services (AWS) [9] caused an outage affecting most North American data centers, knocking offline popular third-party services. Most recently, another incident happened during the peak hours of December 14^{th,} 2020, as an internal storage quota issue in Google Cloud infrastructure [10] caused chaos in Europe. Major Google services like Gmail and YouTube were inaccessible for hours. However, most significantly, most of the third-

party vendors utilizing Google for Single Sign-On (SSO) authentication were unable to operate.

Besides capacity and availability issues, centralization has also taken its toll most significantly on users' data, privacy, and security. All data from those tracking their activity to those comprising their digital identity are often collected and stored in corporate data centers. A data breach at Equifax in 2017 [11] revealed the security risks associated with user details stored in a specific location. In 2013, many top-secret documents disclosed by ex-NSA contractor Edward Snowden revealed shocking evidence about mass surveillance projects by profit data-controlling entities on a global scale [12]. At the same period, another data scandal broke out related to Facebook. Millions of personal profiles were collected and analyzed without user consent by the British political consulting firm Cambridge Analytica toe used in political advertising [13].

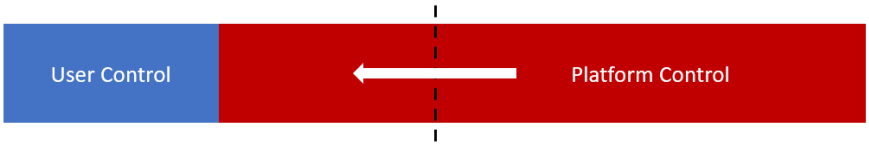


Figure 3 Platforms increasing their control over user information

The examples presented above are just the peak of the iceberg regarding the course the web has taken during its evolution in the 2010s. The bigger picture is drawn around the lack of control over information in a disrupted balance between the users and the platforms, as depicted in Figure 3. Bruce Sterling quite accurately outlined this back in 2012 in his annual conversation with Jon Lebkowsky on The WELL about the state of the world:

> In 2012 it made less and less sense to talk about "the Internet," "the PC business," "telephones," "Silicon Valley," or "the media," and much more sense to just study Google, Apple, Facebook, Amazon, and Microsoft. These big five American vertically organized silos are re-making the world in their image. [14]

This is indeed a very troubling fact. A few large platforms guide the most traffic to online news sources and have enormous influence over what sources of information the public consumes daily. This could lead to severe problems ranging from censorship by the national government to less obvious or unpremeditated favoring the content users see

based on experimental, unaudited recommendation system algorithms integrated into those consolidated platforms.

1.5 Web 3.0, the stateful web

Apparent from the previous paragraph, an alternative route to centralization appears mandatory. The solution may lay in a different approach in the networking model itself.

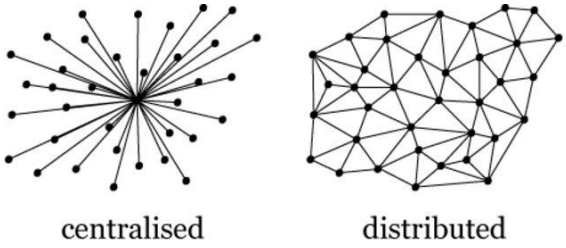


Figure 4 Centralized and Distributed Network [70]

The peer-to-peer model is nothing new. Popularized in the late 1990s with the launch of Napster, a pioneer file-sharing software, it became the zeitgeist of internet enthusiasts with the conception of the BitTorrent protocol in the early 2000s. Peer-to-peer, or P2P as it is commonly referred to, is a distributed application architecture based on equally privileged interconnected peers that form a peer-to-peer network. With the absence of a central server, the P2P model tackles the problem at its root. Schematically depicted in [Figure 4], the user can be connected to numerous peers, simultaneously optimizing resource consumption and bandwidth based on location or other physical network factors. P2P networks have also been proved far more resilient. Any of the peers can go offline without disrupting the operation of the network as a whole. In contrast to the client-server model, the more users utilizing a distributed resource in a P2P network, the more available that resource becomes to the rest of the network, offering a solution to today's scalability issues.

The idea of a decentralized web shares the opinion that some of the principles of P2P networking can be applied, apart from file-sharing, to websites and web applications. This concept of web, also named Web 3.0, promotes autonomy and reveals a new stage of progression on digital culture gradually taking hold. For the above vision to become a reality, another technology already known but sidelined for years had to be incorporated.

As a notion further analyzed in the next chapter, the blockchain was presented widely to the public with the release of the Bitcoin peer-to-peer currency in 2009 [15]. This novel peer-to-peer network protocol can be used for peers to agree on the state of a distributed ledger of transactions. The immutability of blocks participating in a chain is achieved by cryptography. Blockchains redefine the data structures of the web and lay the foundation needed for storage and security for P2P distributed applications in a trustless, decentralized way. This kind of decentralized applications, or dapps, is the subject dealt with in the current dissertation.

1.6 Structure of the thesis

This thesis attempts to provide an overview of the nature of the decentralized applications and the hands-on development of CryptoCerts, a document verification platform for academia. Chapter 2 defines the theoretical background required on cryptography, proceeding with key concepts like blockchain, smart contracts, and decentralized applications. In Chapter 3, the current ecosystem for the development of dapps, along with the most widely-used tools. The development methodology of CryptoCerts is followed in Chapter 4, complemented with showcases of the user experience and smart contract code review from the software engineering perspective. Finally, Chapter 5 concludes our thoughts on dapp development and proposes interesting topics for further research.

2 Key Concepts

To understand how decentralized applications achieve their goals, we need to lay the theoretical framework and explain the key concepts enabling this software development.

2.1 Cryptography

Cryptography is the study and practice of developing protocols to render a piece of information secret. While this definition covers the vital part of private communication necessary for various circumstances from the Antiquity to modern days, it is pretty generic. For the reader to grasp the utility of cryptography for modern applications, the notion of an adversary must be incorporated, as it was introduced by Ron Rivest in 1990 [16]. Indeed, the modern application of cryptography includes the practice of keeping secret information away from adversaries.

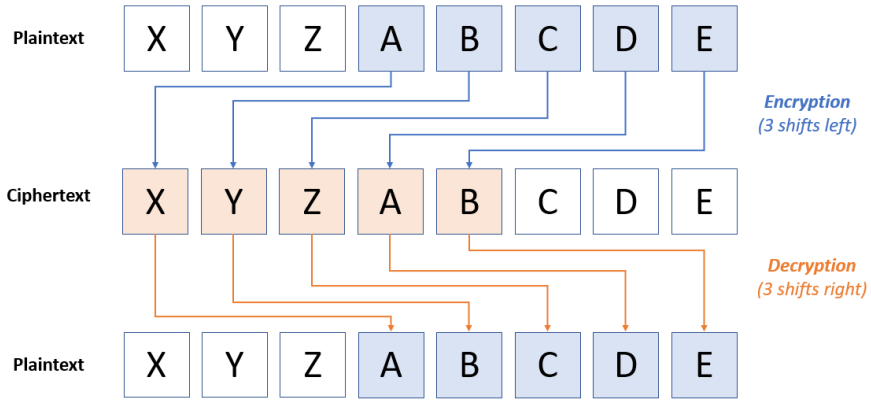


Figure 5 The Caesar cipher

There is a need here also to clarify some of the standard terms used in the cryptography domain. The initial state of a message about to be transmitted is called the plaintext, while the encrypted form is called the ciphertext. The process of converting the plaintext to ciphertext is defined as encryption. On the other hand, the task to transform the ciphertext back to plaintext is described as decryption. The finite sequence of steps that can be followed like a procedure, therefore the algorithm required to encrypt and decrypt a message, is commonly referred to as a cipher. One of the simplest ciphers known from history, referred to as Caesar cipher, was used by Julius Caesar for his private written communication [17, p. 49]. Based on substitution, the cipher involved a simple shift by three positions upwards on the alphabet on the plaintext letters, as depicted in Figure 5. The reverse operation was performed for the decryption.

A key concept differentiating encryption from encoding is the existence of a key. A key is a very accordant term used for a separate piece of information required for encryption and decryption to be performed. If a parallel could be drawn between a cipher and a padlock, the cryptographic key is the actual key that allows its owner to lock and unlock the padlock without a hassle. As the following section will describe, the key is often generated using a specific process. The cipher refers to encryption and decryption algorithms, and the term cryptosystem describes both procedures and the key generation.

The web is an insecure communications system. The reason why the internet today is far more secure than it was in the 1970s and the 1980s lies with the extended usage of cryptography. The generic definition described in the previous paragraph arises because the threats are diverse, and cryptography provides ways to counter them efficiently. When we transmit data online, we need specific properties, and their role is to protect against a particular type of threat. These are:

- **Confidentiality**. The ability to protect from non-authorized disclosure of information. Confidentiality should be performed so that the actual information remains hidden from unauthorized parties, and its very existence shall not be revealed.
- **Integrity**. The protection from unauthorized alteration. The integrity of a message shall be preserved and easily verifiable by the receiver to confirm whether the actual message payload has been modified during the transmission. This also includes inadvertent data corruption due to noise in the transmission channel, often in telecommunications.
- **Authentication**. The assurance that the user or the entity at the other end of the communication channel is who they say they are. Knowing the sender's identity also applies to data authentication, the ability to verify that the party indeed sent the message we believe it was sent from and not an impostor or a spy.
- **Non-repudiation**. The ability to ensure that no parties can claim they did not interact in communication when they did. This includes both non-repudiation of destination: the denial of receiving a message and non-repudiation of origin: the denial of sending a message.

It is essential to clarify that many of the above properties are not achieved separately and more than often overlap with others. Similarly, encrypting a message to

preserve confidentiality also provides integrity, as the decryption algorithm would require the message to be unaltered for the decryption to be successful.

2.1.1 Types of Cryptography

Moving forward to approach the blockchain, we need to describe more the various types of cryptography today. Those are Secret Key Cryptography (SKC), Public Key Cryptography (PKC), and Hash Functions.

2.1.1.1 Secret Key Cryptography (SKC)

Secret Key Cryptography uses a single key to encrypt and decrypt data. Both encryption and decryption utilize the same shared key used by both the sender and the receiver. The scheme depicting a secret key cryptosystem Figure 6 appears symmetrical, so this kind of cryptography is also referred to as symmetric key cryptography.

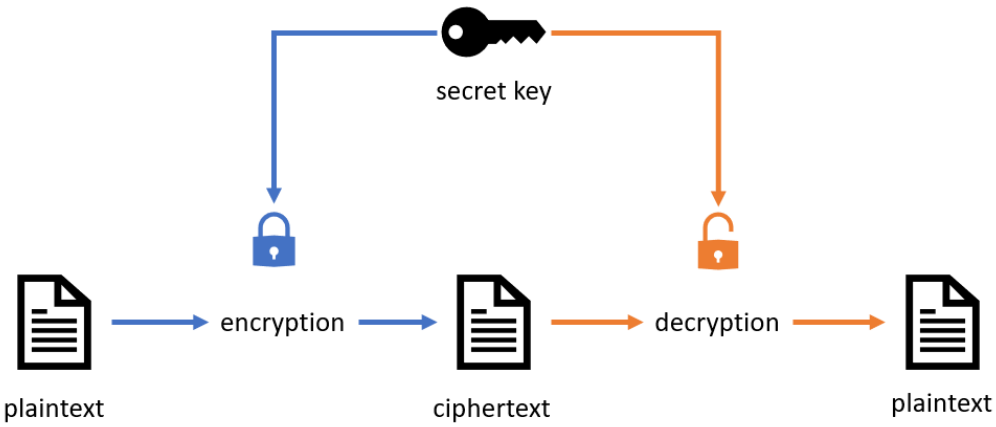


Figure 6 Secret Key Cryptography

The challenge in this type of cryptography is the secure distribution of the secret key. The reason is that the only way to fortify the security of the information in a shared secret key cryptographic system is to protect the secrecy of the secret key. This need requires both participants to communicate through a secure channel before the cryptographic transmission takes place.

Known examples of modern algorithms based on Secret Key Cryptography include the Data Encryption Standard (DES), an algorithm designed by IBM in the 1970s and adopted by the National Bureau of Standards (NBS) in 1977 [18]. The US government moved to its successor, the Advanced Encryption Standard (AES), in 2001 [19].

2.1.1.2 Public Key Cryptography (PKC)

Public Key Cryptography utilizes two mathematically related keys. Knowing one of the keys does not enable someone to determine the other key easily. One key is used to encrypt the plaintext and the other key to decrypt the ciphertext. There is no restriction on which key is applied first, but the importance is that both keys are required in order for the cryptosystem to work. One of the keys is public, and its owner can freely distribute it to the other parties. The other key is private and shall be kept secret and never revealed. The employment of a pair of keys made this type of cryptography also be known as asymmetric cryptography. This asymmetry appears schematically below in Figure 7.

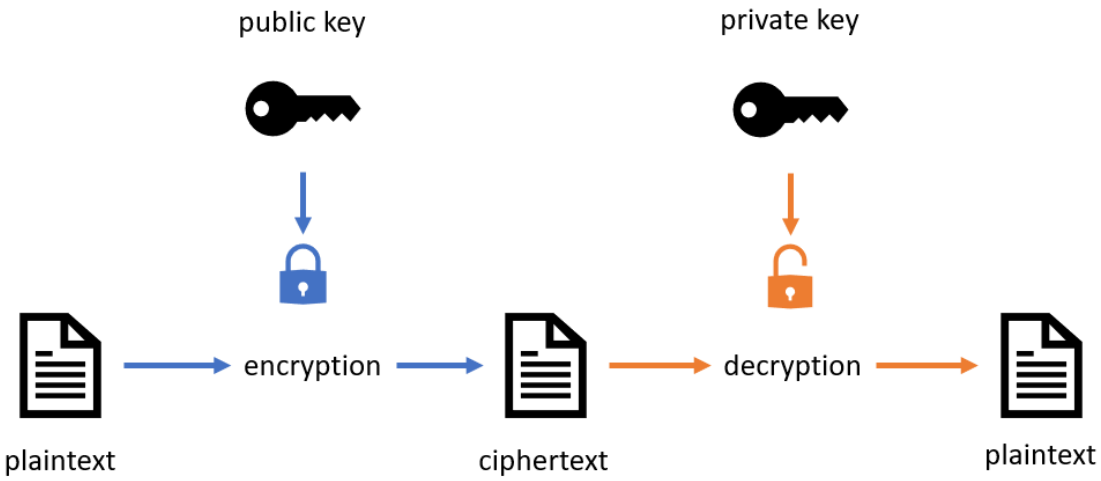


Figure 7 Public Key Cryptography

Public Key Cryptography is considered to be the most significant development in cryptography in the past centuries. It was first described by Martin Hellman and Whitfield Diffie in 1976 [20]. It is primarily used today for authentication, non-repudiation, and key exchange.

A notable application of Public Key Cryptography is the RSA cryptosystem, named out of Ronald Rivest, Adi Shamir, and Leonard Adleman, who invented it [21]. Thousands of software products use RSA for key exchange, digital signatures, or encryption of small data blocks. Other well-regarded asymmetric key techniques for varied purposes include the Diffie Hellman key exchange protocol, the Digital Signature Algorithm (DSA), the ElGamal encryption system, and lots of developments in the field of Elliptic-curve cryptography (ECC).

2.1.1.3 Hash Functions

Hash Functions are algorithms that use no key. They are also known as message digests and one-way encryption. In this approach, one-way mathematical functions generate a fixed-length hash value from the input. The critical aspect is that while the hash functions are relatively easy to calculate, the output hash is very difficult to reproduce the initial input.

Hash functions can map an indeterminable size of plaintext into a fixed size hash, and they are typically used to provide a digital fingerprint of data. A minor change in input will result in a completely different output, as shown in Figure 8. Therefore, many systems employ hash algorithms to ensure the integrity of files or passwords. Hashing is also commonly used with digital signatures, as we will discover in the next section.

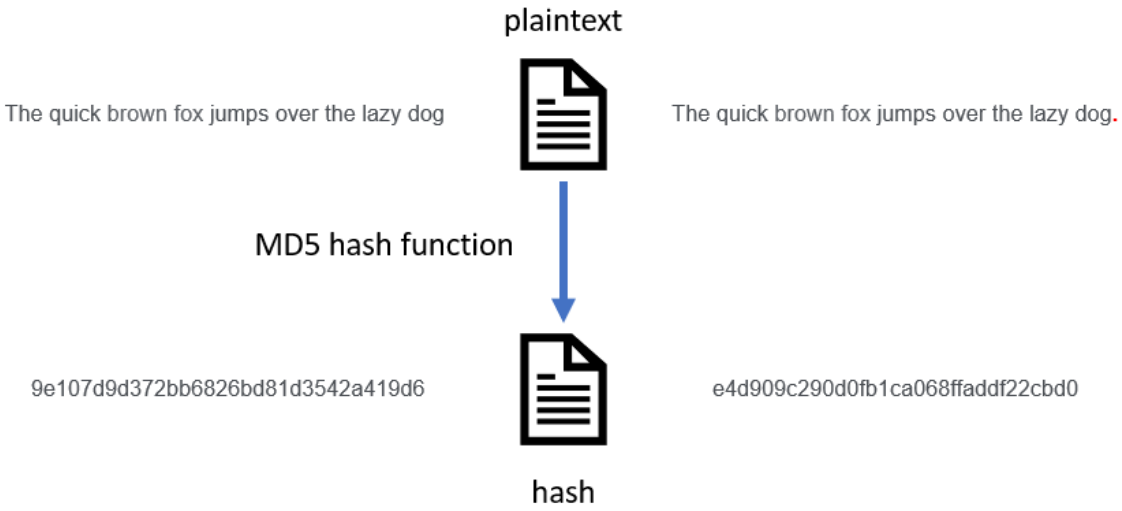


Figure 8 Showcase of MD5 hash function

Famous hash algorithms include the Message Digest (MD) algorithms. However, the latest version, MD5, was identified as cryptographically broken and unsuitable for further use in 2008 [22]. Popular alternatives include the Secure Hash Algorithms (SHA). Although specific variants of SHA-2 are still considered robust and secure, SHA-3 was standardized by The National Institute of Standards and Technology (NIST) in 2015 [23]. SHA-3 is based on Keccak sponge functions.

2.1.2 Digital Signatures

Digital signatures are the public-key fundamentals of message authentication. It is common in the physical world to use handwritten signatures on written messages as they bind someone with a message. Equivalently, a digital signature is a method that binds a

person to digital data. This binding can be verified by the recipient of the message or any other party.

In cryptography, a digital signature is a value calculated from the message and a secret key known only by the sender. Consequently, a digital signature scheme is heavily based on Public Key Cryptography (PKC). Such a scheme can be depicted in Figure 9.

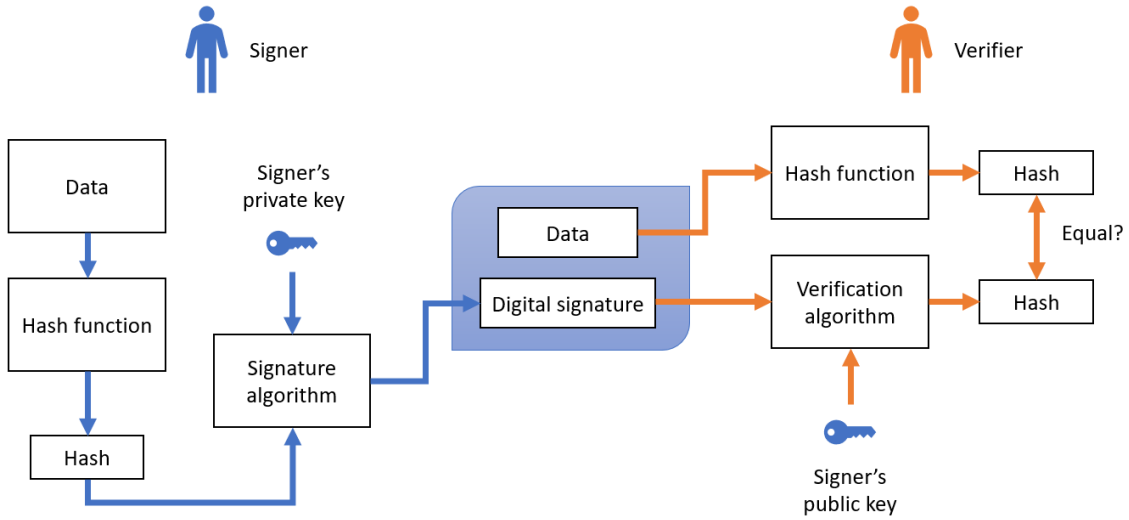


Figure 9 A digital signature scheme

Here are the process steps in detail:

1. The Signer passes data to a hash function and generates a hash.
2. The hash value is passed as input to the signature algorithm along with the Signer's private key. The algorithm produces the digital signature for the given hash.
3. The signature is attached to the data and sent to the Verifier.
4. The Verifier uses the digital signature as input to the verification algorithm along with Signer's public key. The verification algorithm produces an output.
5. The Verifier also passes the received data to the same hash function and generates a hash.
6. The verification relies on whether the hash value and the output of the verification algorithm are equal.

An observant reader would notice that a hash of the data was created instead of signing the original data using the signature algorithm. This decision lies in the efficiency of the scheme. As the data verified can grow quite large on many occasions, signing

using a signing algorithm could be proved computationally expensive and slow. On the contrary, hash values are relatively small in length while guaranteeing data integrity. So, encrypting a hash is far more efficient than encrypting the original data.

2.2 Blockchain

After we have covered the underlying fundamentals of cryptography and cryptographic concepts, we can proceed to a significant notion of this dissertation. Blockchain, a disruptive technology, is considered the basis for providing any reliable and secure decentralized solution.

The notion of blockchain dates back to 1991, when a cryptographically secured chain of blocks was described for the first time by S. Haber and W.S Stornetta. [24] Seven years later, computer scientist N. Szabo worked on a project named "bit gold." Bit gold was about a decentralized digital currency that incorporated property chains powered by solving cryptographic puzzles. The project never reached maturity as it ran into a "double-spending problem," proving it unusable for its cause. [25]

In 2008, at the origin of a financial crisis, a project that would disrupt digital history and bring blockchain back to the surface came into existence. A pseudonymous author named Satoshi Nakamoto submitted a paper to a cryptography mailing list titled "Bitcoin: A Peer-to-Peer Electronic Cash System." [15] The paper laid out the schema for a peer-to-peer network that would support a "system for electronic transactions without relying on trust." Bitcoin incorporates a chain of digital signatures along with a proof-of-work system. The network avoids the double-spending issue thanks to the incentives provided to its peers. Shortly after the dissemination of the paper, the Bitcoin blockchain came into being with the genesis block mined on January 3rd, 2009.

Moving away from cryptocurrencies, we will define a blockchain as a distributed database or a public ledger of records or transactions managed by a peer-to-peer network based on a consensus protocol. [26] The data is recorded in blocks of valid transactions, and the ledger is shared and available to all nodes. The network operates in a trustless way which eliminates the need for a central authority. The identity of participants is verified with the utilization of cryptographic algorithms.

A defining feature of blockchain is immutability. Compared to a regular database that supports the CRUD (Create – Read – Update – Delete) operations, blockchain supports only the CR part. Each block in the ledger cannot be erased after the verification

from the majority of the participants. The history of transactions is permanent and unalterable. The former characteristic also enables traceability, meaning any transaction can be traced back to its verified owner. Blockchains also own a part of their reputation to the reliability of the peer-to-peer network. The ledger is distributed and replicated in its entirety on every node. Therefore, if one or more of them are pushed offline, the integrity of the records is still preserved. All blockchain operations, from identity verification to the addition and verification of new blocks, are based on cryptography. Public key cryptography, hashing, and digital signatures are utilized [27]. These underlying mechanisms make blockchain a highly secure type of technology.

2.2.1 Block structure

As the name discloses, blockchain consists of a sequence of blocks. How each

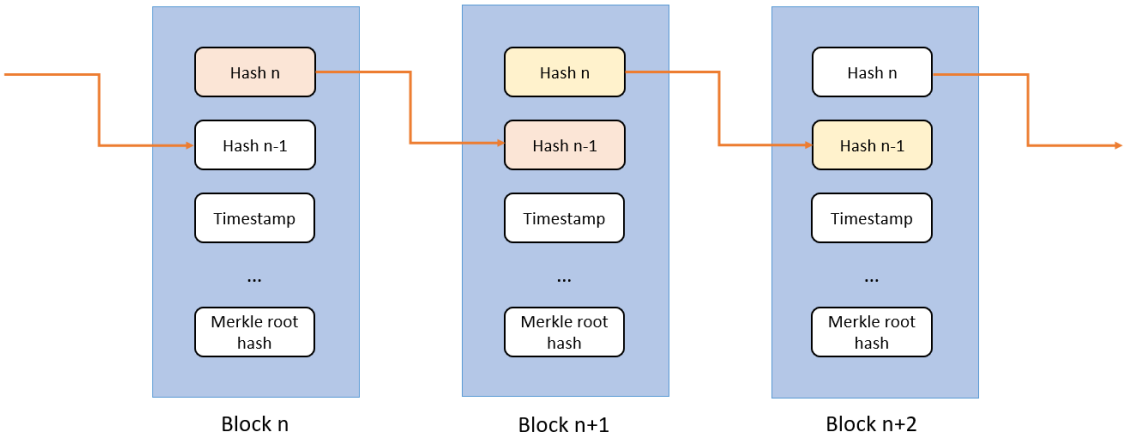


Figure 10 Basic blockchain structure

block is structured can reflect the integrity of this type of storage. Depending on the exact design of the block, the details it includes may vary. A generic blockchain structure can be depicted in Figure 10.

Each block consists of a header and a representation of the transactions it includes. The header can include the current block hash, the hash of the previous block, and a timestamp. As discussed in a previous section, hashing functions preserve the data's integrity while keeping the output to a fixed length. We create an immutable chain by hashing the block and including its checksum into the next block. Any alteration at any part of a block would result in an altered hash and, consequently, an invalid chain.

Blocks also include sets of transactions that are hashed and encoded into a Merkle tree. A Merkle tree is also known as a binary hash tree. It is a data structure that can store

large data sets that can be summarized and verified efficiently using cryptographic hashes. The most common hash functions used in first-generation blockchains are SHA-2, like the SHA-256 variant Bitcoin blockchain uses [15].

2.2.2 Block creation

New blocks are produced repeatedly in the distributed network. Every participating node can submit its transactions to a shared pool, but only those who can get into the next block are appended to the chain. The act of creating a new block in a blockchain network and getting it signed as the next block of the chain is called mining, and the peers who participate in this process are known as miners.

Mining is part of blockchains' security process that creates a competitive environment where all miners are trying to validate data and check that everyone else who has mined a block has done so correctly. Therefore, validating blocks are often accompanied by some reward from the network. Additionally, it is not uncommon for participants to include a fee in their transactions to incentivize miners to prioritize them over others in constructing the next block. For the peer-to-peer network to operate without trusted intermediaries, the process of maintaining the distributed ledger must generate enough incentive for attracting miners. Network rewards and transaction fees serve as incentives, and blockchain protocols typically distribute them using a native token. [28]

Depending on how the consensus is established on a given network, the mining process can differ significantly from model to model.

2.2.3 Consensus models

The mechanisms that dictate how a decentralized, trustless blockchain network reaches an agreement are called consensus models. More often, this agreement refers to which block will be appended next to the blockchain. There are many models developed over the years like the Proof of Work (PoW), the Proof of Stake (PoS), the Proof of Authority (PoA), and the Delegated Proof of Stake (DPoS). Following, we are going to describe the first two as the most famous examples.

2.2.3.1 Proof of Work (PoW)

The Proof of Work (PoW) model is the most prominent example of consensus mechanisms. Hal Finney first introduced the Proof of Work model in 2004 for securing

digital money through the idea of “reusable proof of work” using the SHA hashing algorithm. It describes a process that allows a party to prove that a significant amount of computational effort has been expended. Any observer should be able to verify the act of work with minimal effort.

A typical form of Proof of Work includes the repeatedly hashing of a given input. At the same time, the miner attaches a random nonce to it to calculate a checksum with the requested characteristics. For example, with ‘blockchain’ as an input to the SHA-256 hash function, we look for a ciphertext that should start with five leading zeroes (‘000000’). In order to solve this puzzle, there is no way around it other than use brute force. A daring miner should start by concatenating the initial string with zero, thus ‘blockchain0’, compute the checksum, and check its validity. If it does not comply with the requirement, he must increment the nonce by one and make another attempt.

blockchain0 → bd4824d8ee63fc82392a6441444166d22ed84eaa6dab11d4923075975acab938
blockchain1 → db0b9c1cb5e9c680dfff7482f1a8efad0e786f41b6b89a758fb26d9e223e0a10
blockchain2 → 3108428714518b4a2519e37ff8dce64d37c6ff9d8916cc6596391165db52c2b8
…
blockchain1041 → 00007f73e777e83b01302b5fd5bc9905960c6398c7b24d0f2cc6a3e0c5cd3522

This process will require 1042 attempts, but eventually, a valid output will be generated. In an actual application of Proof of Work, the requested hash could include 18 or more leading zeroes, which would take a significant amount of iterations. However, it becomes evident that regardless of the computing power required to produce a valid hash, the act of verifying it remains as trivial as passing the indicated solution to the hash function and check the result. [29]

Mining does not validate transactions in systems that operate using the Proof of Work as this task is pretty light computationally. It is used for building a reliable commitment against an attack instead. As more blocks are attached to the chain over time, the sum of computing power required to tamper them becomes too cumbersome for an attacker to undertake. Consequently, the actual state of consensus becomes more robust as time passes by. If an attacker wanted to reverse or forge a transaction that happened a couple of blocks in the past, it would have to surpass the current growth rate of the blockchain and recompute all the blocks added after the altered block. So, it is

apparent that all Proof of Work blockchains remain secure as long there is enough computing power supporting them in mining. [30]

Consequently, Proof of Work receives lots of criticism as the hashing power it requires for a mainstream blockchain network to operate leads to significant energy consumption.

2.2.3.2 Proof of Stake (PoS)

The Proof of Stake (PoS) model is a promising consensus mechanism often suggested in place of Proof of Work. Miners in Proof of Stake are called validators. In this model, participants put up their balance of native blockchain tokens as collateral. In return, they get authorization over new blocks in proportion to the amount they stake. Validators are chosen randomly to create blocks and are responsible for validating and confirming blocks they do not create. The act of validation blocks is often referred to as attestation.

In essence, validators get additional authority from ownership over time. As miners, they get rewarded for creating new blocks and attesting to blocks others created. A peer's stake also acts as an additional incentive to encourage good validator behavior. An unreliable validator can lose a portion of their stake for failing to validate blocks or even their entire stake if it attempts to perform as a bad actor. Attesting to malicious blocks can also lead to loss of collateral. [31]

In contrast to Proof of Work, validators do not need significant computing power to perform their duties as they are randomly selected and not competing with each other. Proof of Stake reinstates energy efficiency and lifts any hardware limitations, allowing a broader audience of peers to participate in consensus. Proof of Stake should lead to a stronger immunity against centralization in blockchain networks where token distribution is allocated evenly. There are also indications that Proof of Stake networks could deal with scaling issues more efficiently using techniques like sharding.

2.2.4 Blockchain types

Blockchains can be classified into different categories based on data accessibility, participation authorization, and core functionality with smart contract support. [32]

2.2.4.1 Accessibility

Based on data access, we can identify the following blockchain types:

- **Public**: In this type, anyone can read and submit transactions providing a pair of keys. It is typical for the protocol and code to be open-sourced and maintained by an open community. The network operates in a trustless way. Examples of public blockchain networks are Bitcoin and Ethereum.
- **Private**: In this type, the blockchain is tightly controlled and established between trusted entities. A person or an organization operates the network. In this case, nobody can create new blocks or transactions on the network, verify the ledger history or participate in the consensus without explicit authorization from the owner or the central administrator. Blockchains in this category are based on solid trust to the owner and the participants of his choosing.
- **Consortium or Federated**: In this type, multiple groups of organizations form a consortium and are the only ones allowed to submit transactions and read from the shared ledger. This model attempts to balance public and private blockchains in trust and efficiency and is often seen as partly decentralized. Examples of consortium blockchains are Hyperledger and R3CEV.
- **Hybrid**: In this new type belong networks that combine public, private, or consortium blockchains to streamline transactions. It is another attempt to combine the best of both worlds by keeping information confidential and verifiable by allowing access through a smart contract.

2.2.4.2 Participation

Based on the need for authorization to participate, blockchains can be divided into the following categories:

- **Permissionless**: In this type, the ledger is shared with anyone to create new blocks without permission from a central authority. Participation in the network and blocks verification is allowed to everyone and is achieved using their computing resources.
- **Permissioned**: In this type, participation is only permitted after explicit authorization from an authority. There are cases of permissioned blockchains in which the access to transaction history or the creation of new blocks can also be limited to specific participants. Code distribution

can be open or closed source. In permissioned blockchains, consensus mechanisms can be nonexistent as the network operates on trust between peers. This aspect makes networks of this type high-speed and efficient. In case of malicious activity, the central authority can ban the bad actor and roll back any unpermitted changes. However, permissioned blockchains are considered to sacrifice decentralization in favor of performance.

- **Hybrid**: There is also a possibility of a hybrid model between permissionless and permissioned blockchains. In this type, a node is participating in a permissionless as well as a permissioned blockchain to achieve inter-blockchain communication. There might be cases of a single network to be configured to support both permissioned and permissionless operations.

2.2.4.3 Core functionality

Blockchains can be placed in the following categories based on the core functionality and the smart contract support:

- **Stateless**: In this type, the blockchain network focus is chain functionality and transaction optimization. A new peer joins the networks can obtain the current state from the other nodes in the network. Each transaction can be validated using self-included information. An example of a stateless blockchain network is Bitcoin. The networks supporting similar core functionality are also considered first-generation blockchains or blockchain 1.0.
- **Stateful**: In this type, the networks provide smart contract support and transaction computing capabilities. Stateful blockchains introduce logic to optimize and protect their state and, they are often considered blockchain platforms. A blockchain of this category can support a wide variety of applications by being programmed.

2.3 Smart contracts

The next generation of blockchain technology has brought back a notion that incorporated smart contracts that Szabo first envisioned in 1994. [33] There is no widely accepted definition for smart contracts, but their functionality can be described as agreeing on contingencies derived from a decentralized consensus with low resource

automated execution. Lin William Cong and Zhiguo He attempt to generally define a smart contract as:

> *Smart contracts are digital contracts allowing terms contingent on decentralized consensus that are tamper-proof and typically self-enforcing through automated execution.* [34, p. 9]

Smart contracts are not digital contracts that rely on a trusted intermediary or authority to define consensus and supervise execution. They are also not incorporating artificial intelligence, as the name may imply. Smart contracts can be programmed. They can be thought of as autonomous software programs that, when triggered, execute automatically and compulsorily the rules and conditions set during their implementation. They may aim to enforce an agreement or fulfill a legal contract, execute a transaction, verify or streamline a process. [35]

Smart contracts based their modern existence on blockchain technology. Szabo envisioned that the conception of smart contracts would require the use of cryptography to secure the agreements. However, he did not have any means to guarantee enforcement or transfer of value. The execution of smart contracts on the blockchain ensures that the contract's rules and conditions are tamper-proof. The execution of each contract is recorded as an immutable transaction stored in the blockchain. In turn, smart contract theory appears to lay the foundation of software development on blockchain and makes decentralized applications a reality. They take the static ledger and transform it into a dynamic system able to execute business logic.

2.3.1 Properties

In an attempt to identify a smart contract accurately, the following characteristics can be extracted:

- **Smart contracts are deterministic**. This property means that if a contract distributed in various copies on several nodes receives a specific input, the same output shall be expected on all nodes. Therefore, a smart contract must not include any aspect of randomness or be affected by execution time.

- **Smart contracts are immutable**. This characteristic is the reason why smart contracts need to be executed on a blockchain. Blockchain enforces immutability. As soon as a contract is deployed on the public ledger, it cannot be modified. This aspect is mandatory for trust but introduces some challenges in case of bugs in the code.
- **Smart contracts are verifiable**. When a contract is being deployed on the blockchain, it receives a unique address. Any affected parties can use the address to locate the contract and verify its code and contents before proceeding with its execution.
- **Smart contracts are executed in a limited context**. This property means the contract can access their state, the context of the transaction that called them, and some information about the most recent blocks.
- **Smart contracts are decentralized**. A contract is executed on the local blockchain instance of a node, but as all instances share the same initial state, they deterministically produce the same final state. Thus the whole network operates as a single decentralized world computer.

2.3.2 Oracles

The rules and conditions in a smart contract determine when an action should be executed. When a particular value or combination of values has been achieved, the smart contract is being executed. Its execution, based on preconfigured algorithmic steps, performs a series of operations, resulting in a new state or the emission of an event. The result of the execution is written in the blockchain along with the effect on the participating parties.

There might be cases when the value or the condition required is outside the network boundaries the contract is executed. This situation leads to a problem as it dramatically limits the ability of smart contracts to handle real-world problems. A solution is provided by entities called oracles. In the blockchain and smart contracts context, an oracle is an agent that locates and verifies real-world events and transmits them as data in the blockchain. The primary responsibility of oracles is to provide this kind of data to smart contracts in a secure and trusted way. Such data could be the value

of temperature, a successful money transaction, the increase of a stock price, or the inflation percentage.

There are many types of oracles, depending on how they are being utilized. Software oracles can handle information that can be found online on the web. The sources of the information may vary and could include company or organization websites. These oracles fetch the data from its online resources and forward them to any subscribed smart contract. Other smart contracts require information directly from the physical world. For example, a smart contract could require values from a sensor located at a particular location. The greatest challenge with hardware oracles is their ability to report measurements without sacrificing data security. Oracles can also be classified based on the direction of the information. Inbound oracles can transmit information to smart contracts from external sources. Similarly, outbound oracles can send information to external world interfaces from smart contracts.

2.4 Decentralized applications

A decentralized application (dapp) is built on a decentralized network that combines a smart contract and a frontend user interface. In contrast to web applications, dapps do not rely on a regular backend server or a single computer to run, but they are executed on the nodes of a peer-to-peer network. The frontend part of a dapp can be written in any language, just like an ordinary application, and make calls to its backend.

As covered in the previous paragraph, a smart contract is a piece of software that exists on the blockchain and executes precisely as programmed. Once a contract is being deployed to the network, there is no way to change it. Decentralized applications derive their nature from their backend components. They are controlled by logic written into the contract and not by an individual. This property is also a challenge as the smart contract powering an application needs to be designed carefully and tested thoroughly.

It must also be noted that decentralized applications existed before the introduction of blockchains. Examples of decentralized applications are BitTorrent and Tor, designed to execute on peer-to-peer networks that are not powered by blockchain.

The difference between a conventional web application and a decentralized application in the context of smart contracts and blockchains can be depicted schematically in [Figure 11]. A client-server application utilizes its frontend application to make HTTP requests to a backend API hosted on centralized servers. At the same

time, a decentralized application uses the same technologies for its client to make calls to the interfaces of smart contracts hosted on a blockchain.

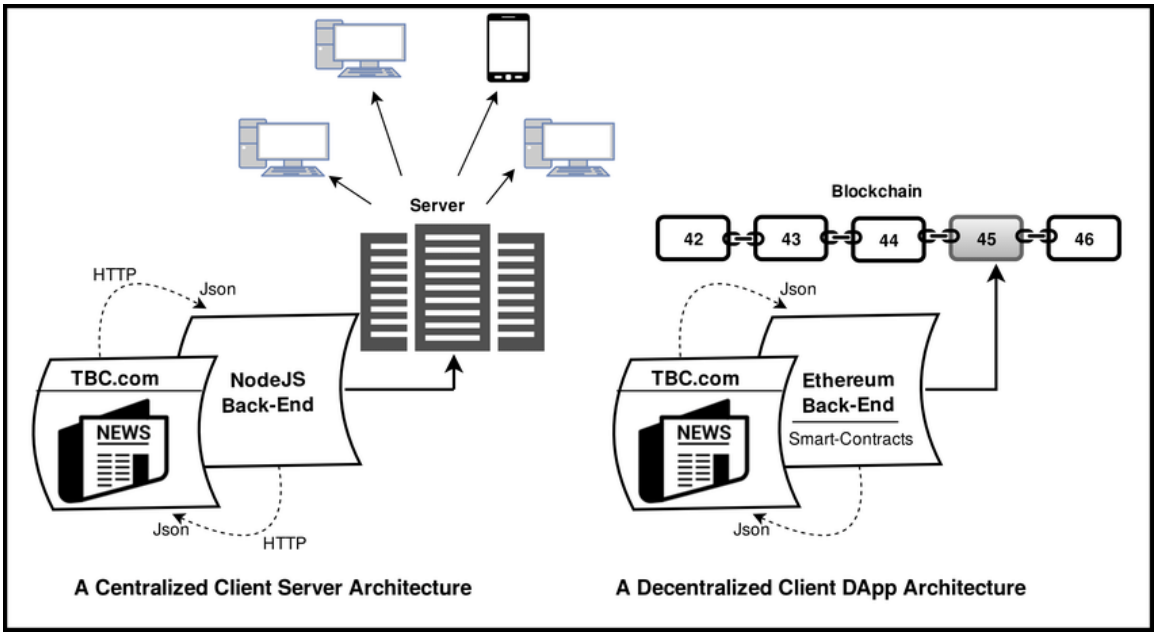


Figure 11 Dapp architecture comparison [71]

2.4.1 Characteristics

The unique nature of decentralized applications inherits some very peculiar characteristics from its blockchain backend. [36] Some of them are listed below:

- They are **decentralized**. These kinds of applications are independent and designed not to be controlled as a group. Their code is distributed to all peers of the network to be executed independently.
- They are **deterministic**. Dapps result at the same output given a specific input, irrespective of the environment they are executed. This characteristic is also a core property of smart contracts.
- They are **Turing-complete**. Based on computability theory, dapps are computationally universal. They are able to perform any action a modern programming language can perform.
- They are **isolated**. Decentralized applications are executed in a virtual environment running on the blockchain network, and therefore, they are fault-tolerant. In case of a bug or a severe malfunction, the application will not prevent the network from normal functioning.

2.4.2 Benefits

Like any traditional app, a dapp's purpose is to provide a solution to a problem. However, decentralized applications development introduces some notable advantages and benefits that typical applications do not.

Decentralized applications offer zero downtime. The smart contract powering a dapp is deployed on every node participating in the network. The network will always be able to respond to clients looking to interact with it. Denial-of-service attacks are not possible against a decentralized application as the peer-to-peer network will mitigate any attempt.

Dapps also provide user privacy. Public key cryptography is used for identity verification against smart contracts and dapps. The users do not need to provide their real-world identity to interact with them.

A significant feature of decentralized applications is censorship resistance. No entity or central authority can block users from deploying or transacting with a dapp or reading data from the blockchain network.

Decentralized applications provide complete data integrity. Any piece of information stored on the blockchain is immutable and irrefutable preserved by cryptographic algorithms. No transactions can be erased, so data cannot be forged.

Trustless transactions are another unique characteristic of dapps. There is no central authority or intermediate required for smart contracts to be analyzed and executed standardly. This behavior may not be the case on centralized applications where the users have to trust the server's owner that will not misuse records, tamper with data, or even get hacked.

Last but not least, decentralized applications are open-sourced. The code executed in the backend in smart contracts is deployed on the blockchain, so there is no way to conceal malicious code. This aspect allows a dapp to be reviewed thoroughly by the open-source community to identify bugs or hidden pitfalls.

2.4.3 Implications

Of course, decentralized applications are not perfect and come with their own set of disadvantages and issues. Next, we will try to identify the most known implications dapps development often encounters.

Decentralized applications are hard to maintain. The code and the data of a dapp are deployed on the blockchain, which makes them hard to modify. Blockchain developers often face difficulties to publish updates to their applications or the data stored by them. Once a dapp is deployed, even bugs and security risks are irreversible in the old version.

Decentralization comes at a price. Decentralized applications deal with serious performance overhead. To benefit from the advantages of blockchains, every node in the network must run and store every transaction. Decentralized consensus also takes its toll. Consequently, there is considerable performance overhead for dapps, and scaling is one of the primary challenges they face today.

Dapps can face network congestion issues. The current model of blockchain platforms provides limited resources. If a dapp is not efficiently using network resources, the entire lattice can be hampered. If transactions are being produced faster than the network's capacity, the pool of unconfirmed transactions can grow significantly, leading to increased transaction fees and delays.

Decentralized applications lack on user-friendly experience. As it is still at its early stages, the dapp ecosystem can be challenging for the average user. There is a stack of tools required to interact with the blockchain in a truly secure way. Key stores are one of the biggest challenges modern decentralized applications face. A lost key can turn the corresponding identity unretrievable.

An often overlooked aspect of decentralized software development has to do centralized implementations. A dapp running on a blockchain does not enforce decentralization on all aspects of the development stack. Many dapps are neutralized by centralized techniques incorporated as the second layer to the blockchain, often to improve user experience. External services store keys or other private information, frontend clients served from centralized servers, or hybrid models that incorporate business logic to be executed on a server before being written to the blockchain are common examples. All the above can eliminate the benefits of decentralized applications running on the blockchain.

Last, decentralized applications cannot handle KYC easily. Many centralized applications rely on user verification which can be easy when a single authority oversees the process. In the dapp model, there is no such entity, and for Know-Your-Customer, verification can be challenging in application domains that require it.

3 Technologies Involved

In this section, we will describe technologies implementing the fundamental theoretical concepts presented up to this point. Many of these projects are pioneers in the domain of decentralized applications and still drive the current developments.

3.1 Ethereum

In 2013, an introductory technical paper was published by a young programmer, Vitalik Buterin. Highly influenced by Satoshi Nakamoto and the Bitcoin network, Buterin reinstates some of the blockchain concepts and combines them with the conceptions of Nick Szabo. He identifies that Bitcoin does support scripting capabilities that could be associated with smart contracts. However, Bitcoin scripting presents several significant limitations, as the absence of a Turing-complete language and the lack of state. Instead of inheriting Bitcoin limitations, Buterin proposes a new protocol named Ethereum for building decentralized applications. [37]

Ethereum incorporates a blockchain with a built-in Turing-complete programming language allowing anyone to write smart contracts and decentralized applications. The smart contract developer can create his own rules for ownership, transactions, and state management functions.

Formal development of Ethereum started in early 2014 with the specification of putting executable smart contracts in the blockchain. The Ethereum Virtual Machine (EVM) is described in the Ethereum yellow paper as the computation engine which acts as a decentralized computer. EVM exists on every node participating in the Ethereum network. It is the part of Ethereum that runs execution and smart contract deployment. A contract is written in a high-level programming language and is compiled into a binary string named bytecode. EVM bytecode is the VM-level machine language that consists of opcodes, operation code instructions for the EVM to execute.

The first release of Ethereum was launched in 2015. The Ethereum Foundation, a non-profit organization, was created. To fund the development, Buterin and other co-founders launched a crowdsourcing campaign to sell participants Ether, the native Ethereum tokens. The public funding round was surprisingly successful, with the Ethereum Foundation raising several million to kickstart operations. Since then, the platform has evolved rapidly, with hundreds of developers involved worldwide.

3.1.1 Ether and Gas

Ethereum includes its native currency named Ether. As mentioned in the whitepaper, Ether:

"*serves the dual purpose of providing a primary liquidity layer to allow for efficient exchange between various types of digital assets and, more importantly, of providing a mechanism for paying transaction fees*" [38].

So, like Bitcoin, Ether is rewarded to miners as an incentive to protect the network security. Additionally, it serves as a mechanism for paying fees per transaction for anti-spam purposes. The smallest denomination of Ether is a Wei, named after the computer scientist Wei Dai. An Ether consists of 10^{18} Wei.

Transaction fees are part of the incentive mechanism in Ethereum as they are in Bitcoin. However, there is a difference in how they are represented and computed. In Bitcoin, the transaction fee is the difference in value between the input and the output of a transaction. On the other hand, fees in Ethereum are not fixed as they are dependent on the code to be executed as part of the transaction. As stated in the whitepaper, a fee exists "*to prevent accidental or hostile infinite loops or other computational wastage in code*." They introduce a limit to the Turing-completeness nature of Ethereum's high-level programming language. While the language is Turing-complete, the developer is limited in the number of steps he can introduce in the implementation. As transactions are executed on every node in the blockchain network, a computational-heavy transaction burdens everyone. This way, fees also act as a measure against spam.

Every Ethereum transaction includes two fields named `STARTGAS` and `GASPRICE`. The `STARTGAS` is the maximum amount of gas the transaction has available to use. The `GASPRICE` is the fee the sender is willing to pay per unit of gas consumed. As a result, the total fee is calculated by multiplying the total gas the transaction used to execute with the `GASPRICE`. If the total gas exceeds the amount of `STARTGAS` provided, then the transaction is reverted.

It is crucial to access that Ethereum manages to decouple the execution cost and the fluctuation in the value of Ether with fiat currencies by providing a separate field for the gas price. This approach is necessary if, for example, the price of Ether increases exponentially on the cryptocurrency market. A transaction with a fixed price per

computation could become prohibitive. This issue is mitigated through the use of `GASPRICE`. In the previous scenario, even if the amount of gas consumed by a particular transaction remains constant, the gas price can be reduced to allow the network to operate at a reasonable cost.

3.1.2 Accounts

The global state of Ethereum is enclosed in account objects that can interact through messages with one another. Each account is identified by an address of 20 bytes and maintains its own state. There are two types of accounts, externally owned accounts (EOA) and contract accounts. Accounts of the first type are controlled by private keys and have no code associated with them. Accounts belonging to contracts are controlled by their contract code and have the relevant code associated with them.

The differences between the two account types are vital in understanding how Ethereum works. An externally owned account can send messages to other externally owned or contract accounts by creating and signing transactions using its private key. A message between two externally owned accounts is a transfer of value between the two. However, the message to a contract account activates the contract's code executing various actions defined by it. Those actions could be the transfer of tokens, the minting of new tokens, a write to storage, a calculation, the creation of a new account, to name a few. It should also be noted that contract accounts cannot initiate new transactions on their own. A contract account can only fire transactions in response to other transactions from either an externally owned account or a contract account. Consequently, any action that impacts the blockchain is always initiated by a transaction controlled by an externally owned account.

3.1.3 Token systems

As Ethereum allows decentralized implementations, it introduces new notions like blockchain tokens. A token is a digital asset that exists on a blockchain but is not a built-in currency. It can be created and owned but also transferred to others, creating economic systems. Token systems are pretty easy to implement in Ethereum. [39]

It is essential to understand that a currency, or token system, fundamentally acts like a database with certain constraints. The database can execute a single operation simulating a transaction. A transaction is performed by subtracting X units from balance A and adding them to balance B. The constraints enforce that balance A had at least X

units before the transaction, and A must approve the transaction. To implement a token system, all that is needed is to implement this logic into a smart contract. [40]

One of the most renowned Ethereum tokens is ERC20, and it has emerged as the technical standard. [41] ERC20 is used as an interface for all smart contracts to implement by providing a list of rules for fungible tokens. Other standards like the ERC721 represent ownership of non-fungible tokens (NFTs), where each token is unique. [42] Various types of NFTs have become popular the recent years leading the frenzy of digital collectibles.

3.2 Solidity

Smart contracts are written in a higher-level language than the bytecode format of the Ethereum Virtual Machine (EVM). Solidity is the most popular smart contract language. As the Ethereum Foundation explains:

> *Solidity is a contract-oriented, high-level language for implementing smart contracts.... It was influenced by C++, Python, and JavaScript and is designed to target the Ethereum Virtual Machine (EVM).... Solidity is statically typed, supports inheritance, libraries, and complex user-defined types, among other features.* [43]

Solidity has reached version 0.8.9 at the time of this writing. However, as version 0.8.x has introduced many breaking changes, numerous libraries did not have the time to adapt, so we relied on version 0.7.0 to implement the current thesis project.

3.2.1 Source file structure

Solidity source files can contain numerous contract definitions, import and pragma directives, struct, enum, function error, and constant definitions. As an example, let us inspect the following Solidity contract in Code 1:

```
1.  // SPDX-License-Identifier: GPL-3.0
2.  pragma solidity >=0.4.16 <0.9.0;
3.
4.  contract SimpleStorage {
5.      uint storedData;
6.
7.      function set(uint x) public {
8.          storedData = x;
9.      }
10.
11.     function get() public view returns (uint) {
12.         return storedData;
13.     }
14. }
15.
```

Code 1 An example contract written in Solidity

At the first line, we encounter the Software Package Data Exchange (SPDX) license identifier. [44] In most circumstances, a contract code will be available as open-source. Every source file is encouraged to include a comment specifying the license under which it was released. The compiler does not validate this tag, but it is included in the contract bytecode metadata.

The `pragma` keyword found on the second line is used to indicate specific compiler features and checks. Every source file should be annotated with a version pragma to prevent incompatible compiler versions from introducing breaking changes to the code. For this example, the `pragma` directive specifies that the source code was written for Solidity versions as early as 0.4.16. At the same time, it supports newer versions up to 0.9.0 but without including it.

The above example does not include one, but Solidity supports import statements following the JavaScript syntax shown in Code 2:

```
1.  import "./filename.sol";
```

Code 2 An import statement

The above statement dictates that all global symbols for the source file specified by the import path should be included in the current file. This statement would be placed below the `pragma` version directive and above the contract definition.

3.2.2 Contract structure

In Solidity, contracts are very similar to classes in object-oriented programming languages. A Solidity contract consists of a collection of code and data. The contract code refers to its functions, while the contract data refers to its state. Every contract

contains multiple declarations of state variables, functions, modifiers, events, errors, struct, and enum types as parts of its structure. [45]

- ***State variables***

State variables are variables that store their values permanently with the contract. The state variables can be modified by transactions executing code that modifies their values. An example of a state variable is the `storedData` variable of the `SimpleStorage` contract above.

- ***Function modifiers***

Functions are very similar to other high-level programming languages. However, function modifiers are unique to Solidity. A function modifier looks like a function but uses the keyword `modifier` instead of the keyword `function`. The modifiers are called before the actual function and thus can change the function's behavior. They are prevalent to provide access control to functions which use should be limited according to specific conditions.

```
1.  // SPDX-License-Identifier: GPL-3.0
2.  pragma solidity >=0.4.22 <0.9.0;
3.
4.  contract Purchase {
5.      address public seller;
6.
7.      modifier onlySeller() { // Modifier
8.          require(
9.              msg.sender == seller,
10.             "Only seller can call this."
11.         );
12.         _;
13.     }
14.
15.     function abort() public view onlySeller { // Modifier usage
16.         // ...
17.     }
18. }
19.
```

Code 3 A function modifier

A function modifier with the name `onlySeller` can be found at line 10 in Code 3 above. The modifier validates whether the contract caller is the seller stored in the state variable using the `require` built-in function. If the validation fails, it will trigger an exception with the description passed as the second argument. The `_;` command will be reached otherwise, meaning the function execution will continue as if the modifier was never applied. The modifier is applied on function `abort()` at line 15.

➢ ***Events***

Events are interfaces to EVM loggers. [46] Applications can subscribe and listen to events through the Ethereum client. When emitted, they are stored as part of the transaction log along with the arguments supplied. If an argument is labeled as `indexed`, it will be converted into a topic and become searchable. An event can have up to three topics. The Log and its event data are not accessible from within contracts, not even from the contract created. An example of a contract event declaration is located at line 5 in Code 4 below. The event is fired using the `emit` keyword inside the `bid()` function on line 9.

```
1.  // SPDX-License-Identifier: GPL-3.0
2.  pragma solidity >=0.4.21 <0.9.0;
3.
4.  contract SimpleAuction {
5.      event HighestBidIncreased(address indexed bidder, uint amount); // Event
6.
7.      function bid() public payable {
8.          // ...
9.          emit HighestBidIncreased(msg.sender, msg.value); // Triggering event
10.     }
11. }
12.
```

Code 4 A smart contract event

3.3 OpenZeppelin

OpenZeppelin is a crypto cybersecurity company providing tools for secure smart contract development and performing security audits for multi-million crypto organizations and platforms. One of its core projects, OpenZeppelin Contracts, is a library for secure smart contract development. The OpenZeppelin library lays a solid foundation for Solidity dapp development by maintaining and extending reusable community-vetted code. [47]

OpenZeppelin smart contracts provide implementations for popular Ethereum standards. Many ERC (Ethereum Request for Comment) tokens are included, like the fungible ERC20 or the non-fungible ERC721 tokens. Another domain that OpenZeppelin supports is smart contract access control. Contracts that provide a general role-based access mechanism or simpler ownership patterns are available. Other provided implementations include utilities like SafeMath that protect against mathematical operations overflows and underflows or multiple proxy patterns that may provide contract upgradeability. [48]

3.4 Truffle

Truffle Suite is an open-source toolbox for decentralized applications development supported by Consensys. The Truffle framework is one of the most popular smart contract development tools that allows compiling, testing, debugging, and deploying any blockchain project powered by the Ethereum Virtual Machine (EVM). [49]

Truffle provides support at all stages of the smart contract development lifecycle. It allows the developer to focus on other parts of the dapp development by handling contract artifacts, library linking, custom deployments, and other complex Ethereum operations. Additionally, it provides a framework for automated contract testing. Smart contract development is pretty fault intolerant, so Truffle assists in building robust code by writing test cases in JavaScript using powerful and proven testing frameworks like Mocha and Chai. Another unique feature of Truffle is the ability to introduce scriptable migration and deployment processes. A complete migration history allows the developer to support any changes that may affect the smart contract code during development or in the future. The framework also handles deployments to support any Ethereum network making smart contracts available on any public or private blockchain. [50]

Truffle comes with its command-line tools like Console and Develop. Both interfaces provide a convenient way to interact with smart contracts for testing and debugging purposes. Truffle Console allows interactive communication with any Ethereum client such as Ganache or geth. On the other hand, Truffle Develop spawns a development blockchain and allows instant communication with any compiled contract.

Truffle is supported by a large community that contributes, among others, to the creation of Truffle boxes. Truffle boxes are preconfigured boilerplate projects, including everything from Solidity contracts to helpful libraries and frontend views.

3.5 Ganache

Ganache is part of the Truffle Suite ecosystem. It is a high-end development tool that allows running a local blockchain for Ethereum and Corda decentralized application development. [51] The local chain is optimized for developing, deploying, and testing projects and smart contracts in a safe and deterministic environment. Utilizing Ganache streamlines the development process that otherwise would require money and time to use real blockchain networks.

Ganache allows the user to fork any Ethereum network without the need to bootstrap a node. It provides a JSON-RPC interface like an actual node. As a versatile development tool, Ganache supports snapshot and revert state mechanisms and fast-forward in time. It also provides the ability to control the creation of new blocks by either choosing instant mining, on-demand, or on a custom interval. Ganache can also impersonate any account for the user without the need to own the private keys.

It comes into two variants: an embedded graphical user interface called Ganache UI and a command-line interface named Ganache CLI. Both versions are available for Windows, MacOS, and Linux.

3.6 Interplanetary File System (IPFS)

The Interplanetary File System (IPFS) is a pure peer-to-peer distributed file system. Like pre-existing peer-to-peer networks, IPFS builds upon the ideas of Distributed Hash Tables (DHTs), BitTorrent, Git, and Self-Certifying File System (SFS). It attempts to simplify and connect successful techniques into a single system. [52]

As with most decentralized technologies, IPFS aims to remove the central points from the architecture, giving the nodes the same capabilities. IPFS nodes store IPFS objects in local storage. They are connected and transfer objects, which can represent files or any other data structure.

IPFS relies on three fundamental principles: content addressing, Directed Acyclic Graphs (DAGs), and Distributed Hash Tables (DHTs).

Content addressing is used to identify content by what is included rather than where it is located. Therefore, every piece of information on the IPFS protocol has a content identifier or CID, essentially the content hash. Using Interplanetary Linked Data (IPLD), which can link between different hash-linked data structures to allow their unification [53], IPFS manages to identify and address content on the network uniquely.

IPFS takes advantage of Directed Acyclic Graphs (DAGs) and specifically Merkle DAGs. This variant of DAGs has each node have a unique identifier that is a hash of the node’s content. [54] This mechanism allows identifying data objects by the value of their hash, which is essentially content addressing. IPFS splits data into blocks to build a Merkle DAG and allow parts of a single file to be more efficiently distributed in the network. This approach also allows duplicate content referenced by the same node and only updated parts to receive new CIDs.

IPFS uses Distributed Hash Tables (DHTs) for content discovery. As a hash table is a database of hash keys to values, a distributed hash table is a table that is split across all peers in a distributed network. [55] In IPFS, a node in search of content has to query the DHT twice. Once to find out which nodes host the blocks comprising the content it is after and once more to locate those nodes through the routing table.

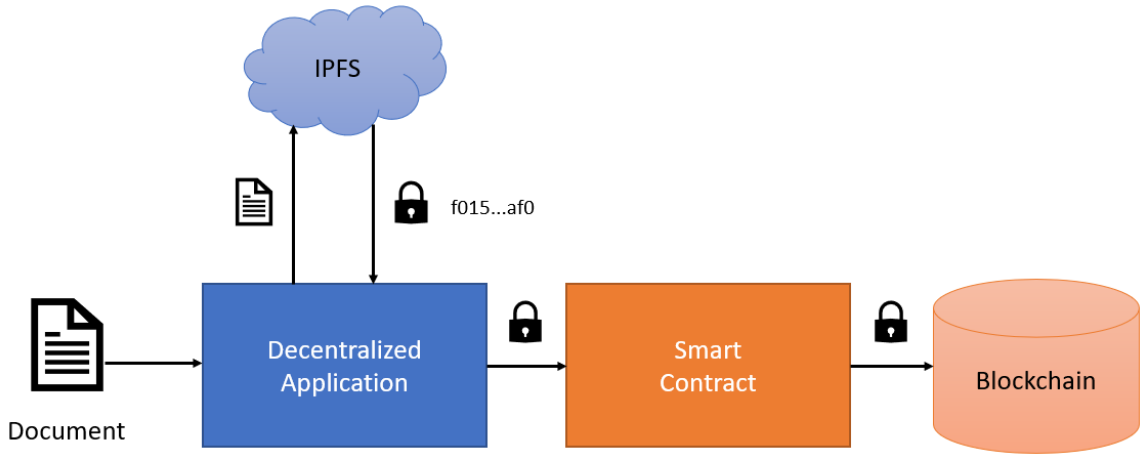


Figure 12 How a dapp can utilize IPFS for document storage

IPFS can empower decentralized applications as it enables them to store large files off-chain and put immutable permanent links using content addressing in the blockchain transactions. This methodology is also followed by CryptoCerts, which utilizes IPFS for storing actual documents, as the example depicted in Figure 12.

3.7 React

React, or React.js is an open-source JavaScript library used to build user interfaces for single-page applications. It is used for handling the view layer in a declarative way for both web and mobile applications. [56]

The framework was first created by Jordan Walke, a software engineer working for Facebook. In 2011 a prototype was used for Facebook's search element, and the following year, Facebook deployed React for its newsfeed. At the same time, the newly acquired Instagram adopted the framework too. In 2013 React gets released as an open-sourced project. [57]

React's philosophy is all about reusable UI components. It focuses on building encapsulated components that manage their state and uses them to make more complex user interfaces. The framework aims to be fast, scalable, and straightforward.

React uses JSX for templating. JSX is an XML-like syntax extension of JavaScript that assists in working with user interfaces inside the code. This feature is

helpful as React components contain both markup and logic in the same file as they are considered inherently coupled. However, the framework is flexible, and JSX is not required in case of preference.

Another unique feature of React is the single-way data flow model. A set of immutable values are passed to the components as properties. The component cannot directly modify any properties. Instead, components are provided callback functions to pass any events from user interaction up to components hierarchy. This process is known as “properties flow down; actions flow up.” A recent addition that allows extracting reusable stateful logic from components not hierarchically related is called React hooks.

React also introduces the virtual DOM (Document Object Model) concept as part of its internals. This mechanism allows the framework to keep a representation of how the user interface should look in memory. When the state has changed, the framework syncs the virtual DOM with the “real” DOM. This process is called reconciliation and allows React to stay declarative.

3.8 Redux

As the requirements for modern web applications grow more complex than before, JavaScript applications must manage more state in code. This necessity introduced tools that make state mutations predictable by imposing restrictions on how and when the update should happen. Such a tool is Redux, a global state manager that often is integrated into React applications. [58]

Redux introduces three fundamental principles to keep stateful logic consistent:

- A single point of truth – The global state is stored within a single store.
- The state is read-only – State changes only by emitting an action.
- The state is changed only by pure functions – The managing functions should be deterministic with no side effects.

In Redux, a pure function that takes action and the current state of the application and returns a new state is called a reducer. An action describes the event that happened, and the reducer needs to determine how the state will be affected, if at all. Last, the central state container is also known as the Redux store. It can be split up into slices that contain reducer logic depending on the features of the application.

3.9 Web3.js

Web3.js is a collection of JavaScript libraries that allow interaction with the Ethereum ecosystem. [59] As a convenience library, web3.js uses a provider to connect to a local or remote Ethereum node. The Ethereum provider JavaScript API is itself very simple and wraps Ethereum JSON-RPC formatted messages. This flow is depicted below in Figure 13.

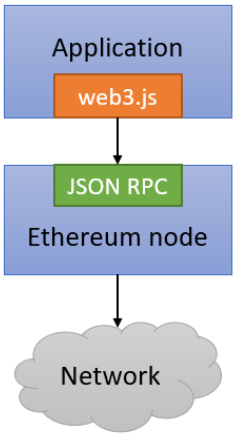


Figure 13 Web3.js to node communication

Web3 provides separate modules for different network functionality. The most popular is `eth`, which consists of tools for the Ethereum blockchain and smart contracts. CryptoCerts client uses web3.js for typical operations like fetching the active network identifier or the connected accounts and interacting with the CryptoCerts smart contract interface. Other modules include the `net` for interacting with network properties and protocol-specific modules like the `shh` focusing on Ethereum's peer-to-peer whisper protocol.

3.10 MetaMask

MetaMask is an Ethereum web browser extension that acts as an Ethereum wallet. [60] More than that, Metamask provides an interface for Ethereum-based decentralized applications; it is a Web3 provider. It essentially acts as a bridge between the blockchain and the browser.

MetaMask allows users to manage accounts and their keys while isolating them from the application context. This approach provides improved security over storing the user keys on a single central server or even in the web browser local storage.

The developer only has to interact with the `window.ethereum` object that gets populated once MetaMask is installed. When a request for a blockchain transaction is being made, MetaMask will prompt the user with all the required details for review.

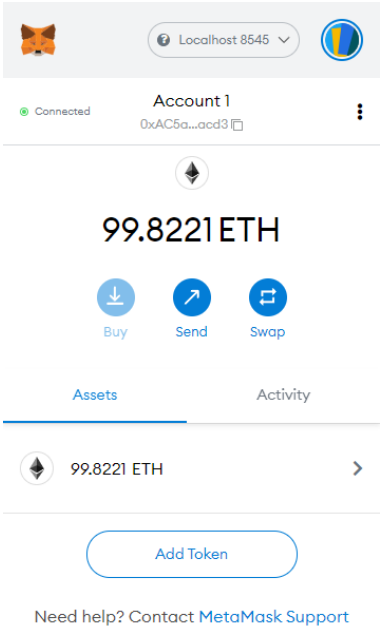


Figure 14 MetaMask connected to localhost

Additionally, MetaMask allows connecting to the main Ethereum blockchain or other test networks without running a node locally. This feature is achieved using an intermediate service named Infura that hosts the nodes and exposes an HTTP API to them. However, this could be considered as a compromise to decentralization. CryptoCert does not utilize Infura and asks MetaMask to connect to a local blockchain network exposed by Ganache, as seen in Figure 14.

3.11 Docker

Docker is an open-source platform that allows run applications using containers. [61] It enables developers to package their code and operating system, libraries, and dependencies required to run in any environment. Docker provides many benefits, including environment isolation, consistency, and portability of code.

Docker is a containerization platform. Containerization provides an alternative to virtualization, but it is essentially virtualization at the operating system level. Containers enable virtualization capabilities into the Linux kernel and allocate resources between multiple applications components of a single host operating system. This process can be parallelized with a hypervisor that makes it possible for multiple virtual machines (VMs)

to share the same CPU, memory, and other resources of a hardware server. However, containerization offers much more cost-effective scalability and improved performance.

Containers existed for many decades, but they took off the ground in 2013 with Docker's emergence. [62] Docker evolved into an industry standard for containers by providing simple tools for developers that accelerated their adoption. The toolkit provides a single API that supports building, deploying, running, updating, and stopping containers with single commands that can be easily automated.

Docker containers are created out of images. A Docker image contains the required source code and all the tools and dependencies to run as a container. A running container is an instance of a Docker image. To compose an image, a developer can write the sequence of commands as instructions for building it in the form of a Dockerfile. There is also the option to use an existing image available on a shared registry like the Docker Hub.

3.11.1 Docker Compose

Among other tools, the Docker ecosystem has introduced Docker Compose. Compose allows the user to create stacks with multiple containers that will reside on the same host. Docker Compose uses a YAML file that allows the developer to configure the containers of his application as services. [63] An example YAML file is shown in Code 5 below. This file is being used to define the backend stack of the CryptoCerts dapp.

```
1.  version: "3.7"
2.
3.  services:
4.    truffle:
5.      container_name: truffle
6.      image: truffle-image
7.      build: .
8.      volumes:
9.        - .:/dapp
10.     entrypoint: sh
11.     command: '-c "sleep 1d"'
12.   ganache-cli:
13.     container_name: ganache-cli
14.     image: trufflesuite/ganache-cli:latest
15.     volumes:
16.       - ganache_db:/data
17.     ports:
18.       - 8545:8545
19.     command: "--db /data -a 10 -d --mnemonic '${MNEMONIC}'"
20.
21. volumes:
22.   ganache_db:
23.
```

Code 5 CryptoCerts backend Docker Compose YAML file

Everyday use cases for Docker Compose include the spin-up of development environments with a single command, automated testing environments as part of CI/CD pipelines, and deployments to single host production servers. For multi-host deployments in clusters, Docker Compose can be combined with the Docker Swarm Mode orchestrator.

4 CryptoCerts

A part of the current dissertation was implementing a fully functional decentralized application to showcase the concepts and the tools presented up to this point. This section is dedicated to giving an overview of the functionality and the implementation details of a prototype dapp named CryptoCerts.

4.1 The problem

Education is a crucial part of our lives, and a degree or a diploma from a recognized institution can increase the chances of a bright career. As a result, an academic certificate has become one of the most sought-after necessities. As assets of great value, academic credentials were inevitably targeted by malicious actors. Fake degrees are a widespread menace in the modern world. According to a report from ACEI Global, forged academic certificates is a billion-dollar industry. [64]

In an attempt to identify the problem, we should emphasize that the traditional certificate system is based on trust. As part of the global employment process, an employer can be identified as the trustor. On the other hand, the trustees are the job candidates, former academic students in our context. When a candidate presents his certificate to an employer, the employer has the option to trust that the certificate is valid. He can trust that the candidate did not participate in any fraudulent activities to produce this certificate. As both the candidate and the employer are not familiar with each other based on initial trust theory, the employer might have to contact the academic institution to authenticate the certificate.

Of course, the trust schema can extend to another relationship between the employer and the academic institution. In that additional schema, the employer trusts the university for the activity in which professors and university officials have assessed that the students have achieved the learning outcomes claimed by the certificate they received. For the sake of simplicity, we assume that academic institutions around the globe are credible and account for trusted authorities.

A proposed solution to the forged credentials problem could be introducing a system to eliminate any need for trust between the transactors. The system will verify the authenticity of the issued certificates by checking the integrity and the validity of the documents directly from the trusted authorities. Blockchain technology can power the

system to ensure the immutability of records and the validity of the transactions through rules coded in smart contracts. This implementation will eliminate the aspect of trust and turn the process transparent and trustless. ACEI Global also referred to blockchain as a technology solution to forged credentials in the past. [65]

4.2 A decentralized academic certificate registry

CryptoCerts is a decentralized academic certificate registry for Web3. The application utilizes the Ethereum blockchain to safeguard the issuance of academic documents, especially certificates and degrees, from educational institutions and organizations to their students. The complete code of its final implementation is referenced in Appendix B.

4.2.1 Entities

For the prototype version of a decentralized registry, we can identify the following user roles as system entities:

- The **Administrator** – He is the sole smart contract owner. He is also responsible for bootstrapping the application by registering the accounts of the Institutions.
- The **Institutions** – They represent the academic institutions or organizations participating in the registry. They issue academic certificates or other documents and assign them to Students.
- The **Students** – They represent the designated students that receive any issued certificates or documents in their name.
- The **Guests** – This is not an explicit role and includes any observer not actively participating in the registry. As the data are all deployed on a public blockchain, anyone can verify them. CryptoCerts provides convenient tools for that purpose.

4.2.2 Features

Next, we will cover the functionality and the features included in CryptoCerts:

- Institutions are able to issue and assign academic certificates to their students in a decentralized and immutable way utilizing smart contracts running on the distributed ledger of the Ethereum blockchain.

- Students can browse the certificates assigned to them and prove ownership by their private key. Using CryptoCerts, they can also fetch a copy of the actual document, stored distributedly in the InterPlanetary File System (IPFS).
- Third-party guests can validate any genuine digital document in their possession against the blockchain records to verify its integrity and authenticity.

Adding rich features that may come across a full-pledged registry platform was never considered in scope for this project. This dissertation aims to provide a proof of concept version, a minimum viable product (MVP).

4.2.3 User stories

CryptoCerts was developed following well-established agile software engineering methodologies. For the initial requirement analysis, the following user stories were composed:

- ✓ *"As an Administrator, I want to be able to create a new Institution by providing its Name, Location, and Account Address."*
- ✓ *"As an Administrator, I want to be able to edit the Title and the Location of an existing Institution."*
- ✓ *"As an Institution, I want to be able to issue a new Certificate by providing its Title, the Student Account Address and by attaching a genuine copy of the actual PDF document."*
- ✓ *"As an Institution, I want to be able to browse through the Certificates my Institutions have already issued."*
- ✓ *"As an Institution, I want to be able to download a copy of the document of any Certificate I have already issued."*
- ✓ *"As an Institution, I want to be able to edit the Title of a Certificate I have already issued."*
- ✓ *"As a Student, I want to be able to browse through all the Certificates already assigned to me from all the Institutions."*
- ✓ *"As a Student, I want to be able to download a copy of the document of any Certificate I have been assigned to."*

- ✓ *"As a guest, I want to be able to browse through the Institutions participating in the registry."*
- ✓ *"As a guest, I want to be able to upload a copy of a document and get a response on its authenticity."*

4.2.4 Wireframes

Based on the features described above in natural language from the end user's perspective, we extracted a series of wireframes.

Wireframes are created to arrange elements to accomplish a particular purpose best. They aim to inform about a business objective or a creative idea. A wireframe depicts an application page layout, including interface elements, navigational systems, and how they work together. The wireframe usually lacks typographic style, color, or graphics, since the main focus lies in functionality, behavior, and priority of content. CryptoCerts wireframes can be found in [Appendix A].

We should also highlight that both the user stories and the presented wireframes are not expected to match the final implementation fully. They are both lightweight methods that allow the development team to identify a product requirement quickly.

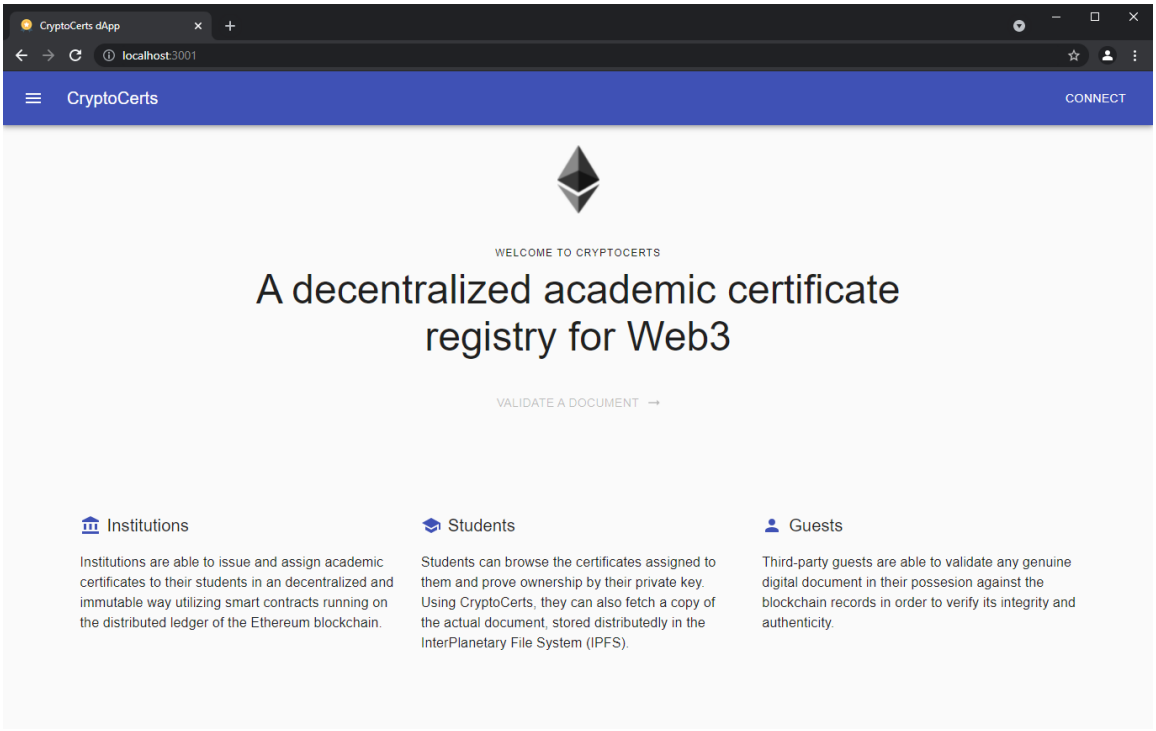
4.3 User experience

As mentioned before, decentralized applications lack user experience. For example, the average user may find it challenging even to navigate an application that requests him to connect a wallet instead of creating an account. Consensys, a leading company in blockchain application development, considers user onboarding one of the obstacles of dapp adoption. They introduced a framework of best practices named Rimble to guide decentralized applications UI/UX guidelines. Rimble provides Web3 components to assist developers in preserving a smooth user experience. [66]

CryptoCerts attempts to adopt some of these principles, guiding the user with new concepts like authentication and transaction signing. In the following paragraphs, we present the basic operations of the decentralized certificate registry from the user viewpoint.

4.3.1 Connecting to the network

Connection requires some technical preparation, and we need to guide new users through it. This process involves helping them use a supported browser, access the right network, and install MetaMask as a Web3 provider in the case of desktop. At the same time, we need to avoid ruining a great experience by overwhelming our users with information as soon as they land on the dapp.



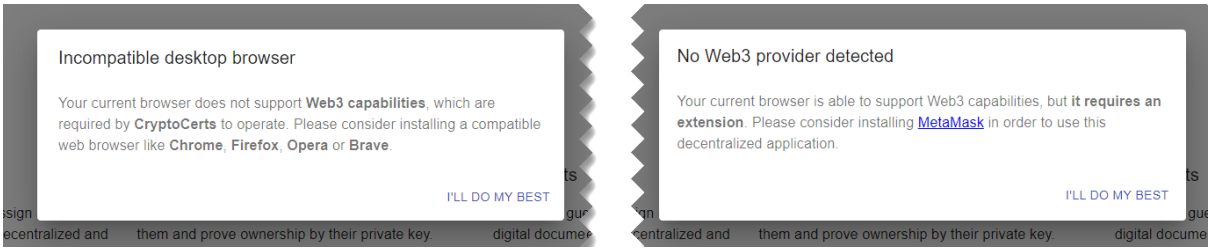
Screenshot 1 CryptoCerts landing page

As seen in Screenshot 1, the CryptoCerts landing page informs the user about the application's features and keeps the experience simple. CryptoCerts acts passively and will let users navigate pages that do not require connection, even using an unsupported browser or connecting to the wrong network.

Screenshot 2 Functionality is limited without a connection

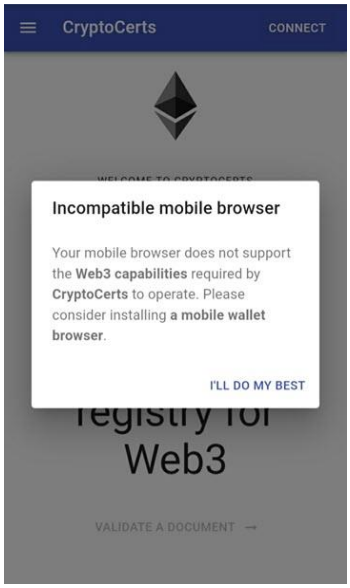
At this point, the vertical navigation bar on the left is available, but only the Home link is active, representing the current page. The "Validate a document" button is deactivated and displays a tooltip on mouse hover that not all capabilities are available. Both can be seen in Screenshot 2. The user is currently a guest using a conventional

browser equipped with no Web3 capabilities. There is no way to provide any blockchain information at this point.



Screenshot 3 Unsupported desktop browser and no Web3 provider found messages

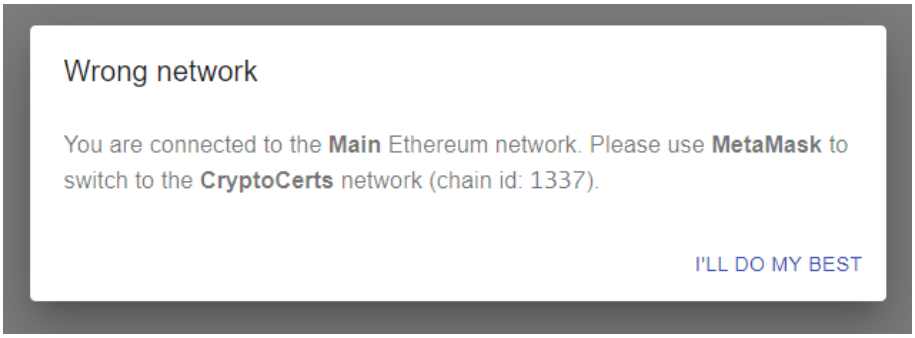
As soon as the user clicks on the "Connect" button in the upper right corner, he will receive more information about the current state. CryptoCerts include educational popup messages to walk the user through the process of using a decentralized application. A message will provide options if the user uses an incompatible browser with no Web3 capabilities. In case the user uses a supported browser but no Web3 provider is available, a message will let him know and suggest he install MetaMask by providing a link. The two cases appear in Screenshot 3 above. Even if the user somehow reaches CryptoCerts from a mobile device, an attempt to connect will detect his setup and guide him appropriately, depicted in Screenshot 4.



Screenshot 4 Incompatible mobile browser message

At this point, the user has to install MetaMask to continue the interaction with the application. Next time he attempts to connect, CryptoCerts will check the active network the Web3 provider is pointing. As this prototype is deployed on a local Ganache instance and not the main Ethereum network, we need to ensure this is also the case for

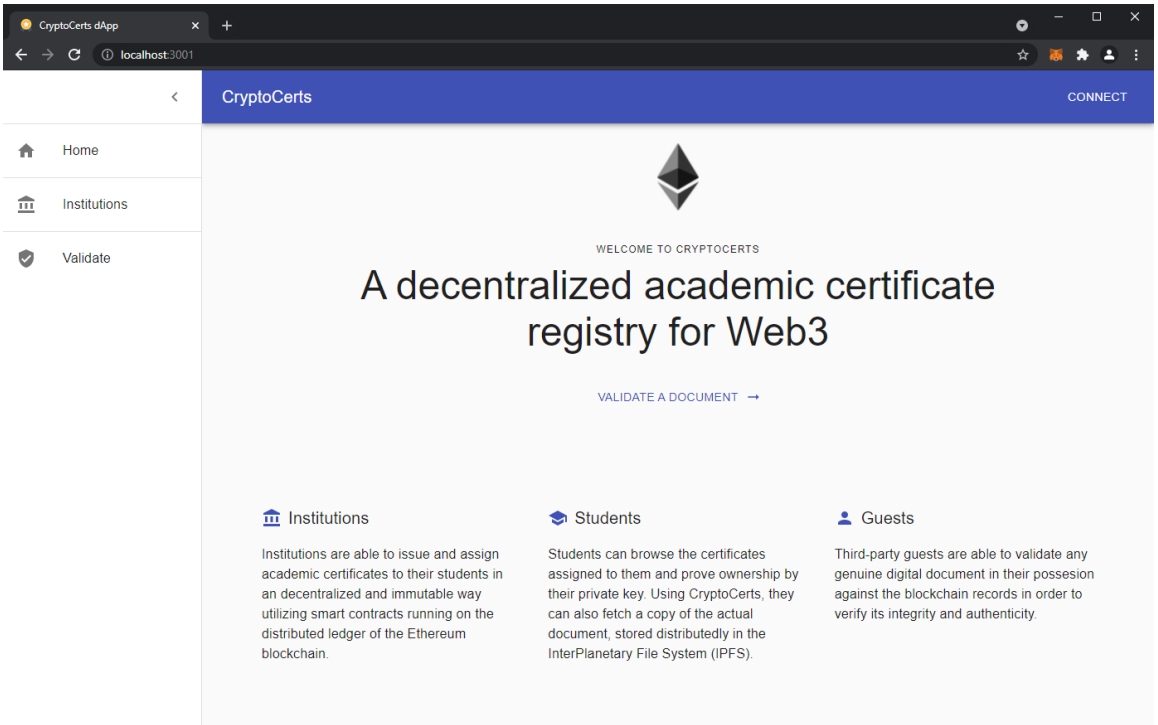
MetaMask. If not, the application notifies the user to switch his network to the correct one, as seen in Screenshot 5.



Screenshot 5 Wrong network is selected message

Eventually, the user will fulfill all the prerequisites required to use a decentralized application. He uses a Web3-compatible browser with a Web3 provider installed that points to the right blockchain network.

At this stage, a new set of functionality unlocks for the user. The application can populate pages with data from the blockchain records. Fulfilling the relevant user stories, a guest user can see a list of the participating institutions and also be able to validate a document against the registry records. Both links are now available in the vertical navigation bar, and the “Validate a document” button is active, as seen in Screenshot 6.



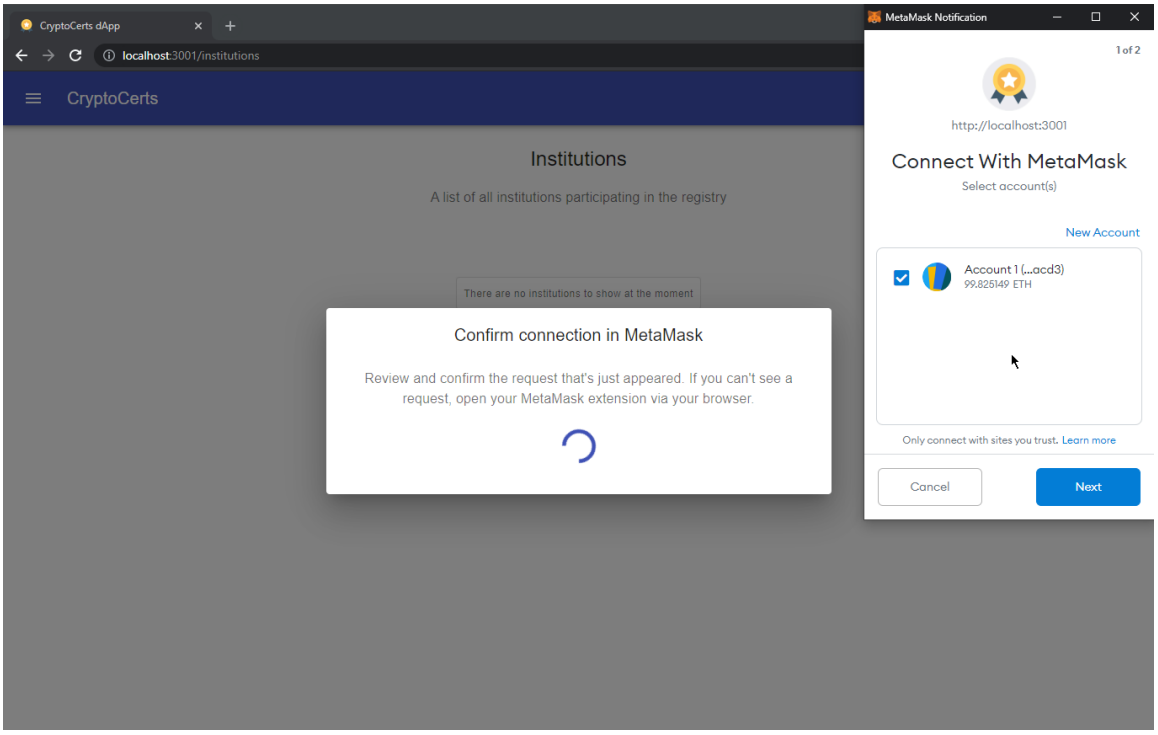
Screenshot 6 Connected to the blockchain

We should clarify that those operations do not require a transaction from an account, so the user does not need to import his keys yet.

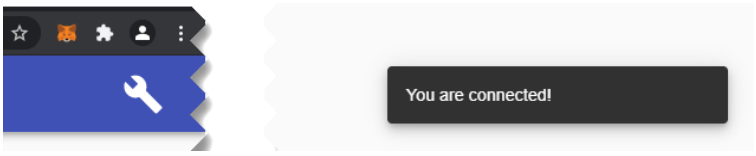
4.3.2 Creating an institution

Let us try to write some records to the blockchain by importing some academic institution accounts. This functionality is only available to the owner of the smart contract, the application administrator. The user has to prove his identity, and the only way to do so in the decentralized web is by providing his keys.

CryptoCerts does not handle user keys directly. This part concerns the Web3 provider, MetaMask, in our case. The user will have to leave the page and complete the operation outside the browser's viewport. Naturally, this may confuse or alert the average user as it exceeds the flow of a conventional web application. Following the best practices, CryptoCerts will provide the required feedback to guide the user.

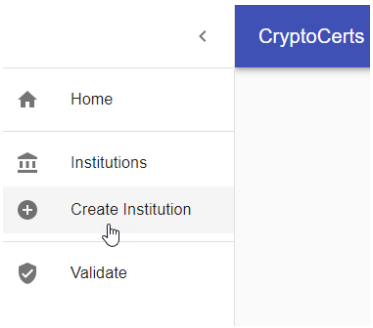


As soon as the user clicks the "Connect" button, several events occur on the screen. MetaMask is triggered by the application and fires a connection notification in the form of a popup window. At the same time, a CryptoCerts popup message with a spinning loader and a set of instructions appears. As the MetaMask connection notification can easily be discarded by miss clicking, the persistent message on the dapp encourages the user to seek the notification and complete the connection.



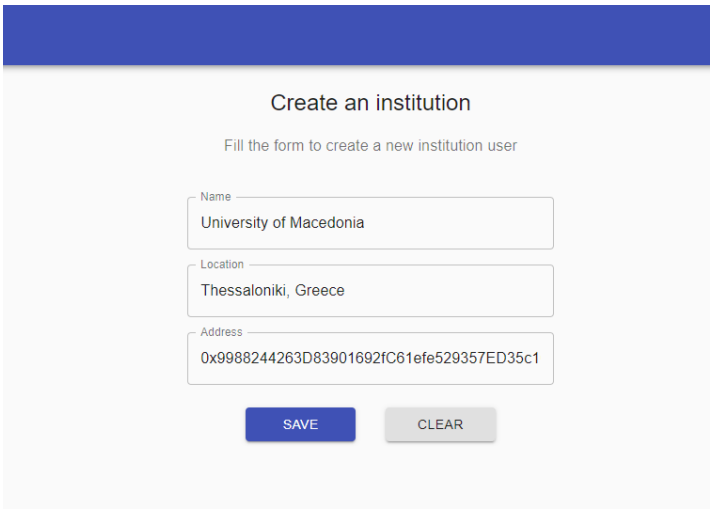
Screenshot 7 The user role icon and the connection notification

The Web3 provider will guide the user to prove his identity either by importing his private key directly or in the form of a BIP32-compatible seed phrase. By using the account associated with the contract owner, CryptoCerts identifies the user as the Administrator. A user menu appears in the top right corner of the screen instead of the connection button. The user role icon appears on it. Additionally, an on-screen notification is triggered on the lower right corner, informing the user that is now connected using an account. Both are depicted in Screenshot 7.



Screenshot 8 Menu updated with Administrator actions

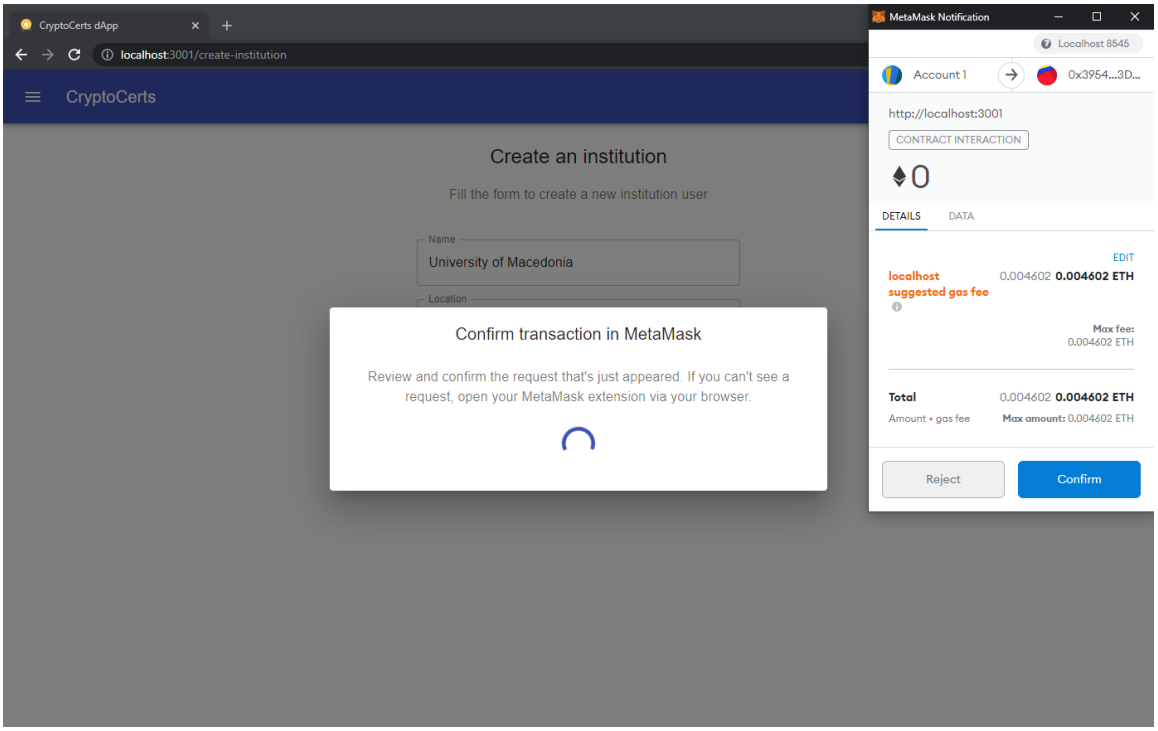
As the application Administrator, the "Create Institution" menu item is now available on the navigation bar, as seen in Screenshot 8. Clicking on it, the user transfers himself to a new page containing the institution creation form. The specific fields required for this demo are the institution name, location, and, most importantly, its externally owned account identifier. The form validates the account to ensure no invalid addresses are assigned. The relevant screen is depicted in Screenshot 9.



Screenshot 9 Filling the institution creation form

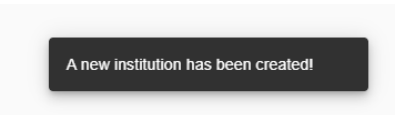
The user is about to perform a transaction. The information will be passed as arguments to the CryptoCerts smart contract, and the result will be written to the blockchain as part of a new block. Another series of events get triggered when the

Administrator clicks the “Save” button to submit the form, as seen in [Screenshot 10]. MetaMask shows another notification requesting the user to confirm a new transaction. The address of the smart contract is displayed along with the gas cost details. Simultaneously, CryptoCerts draws another variant of the spinner popup message on the screen instructing the user to confirm the transaction.



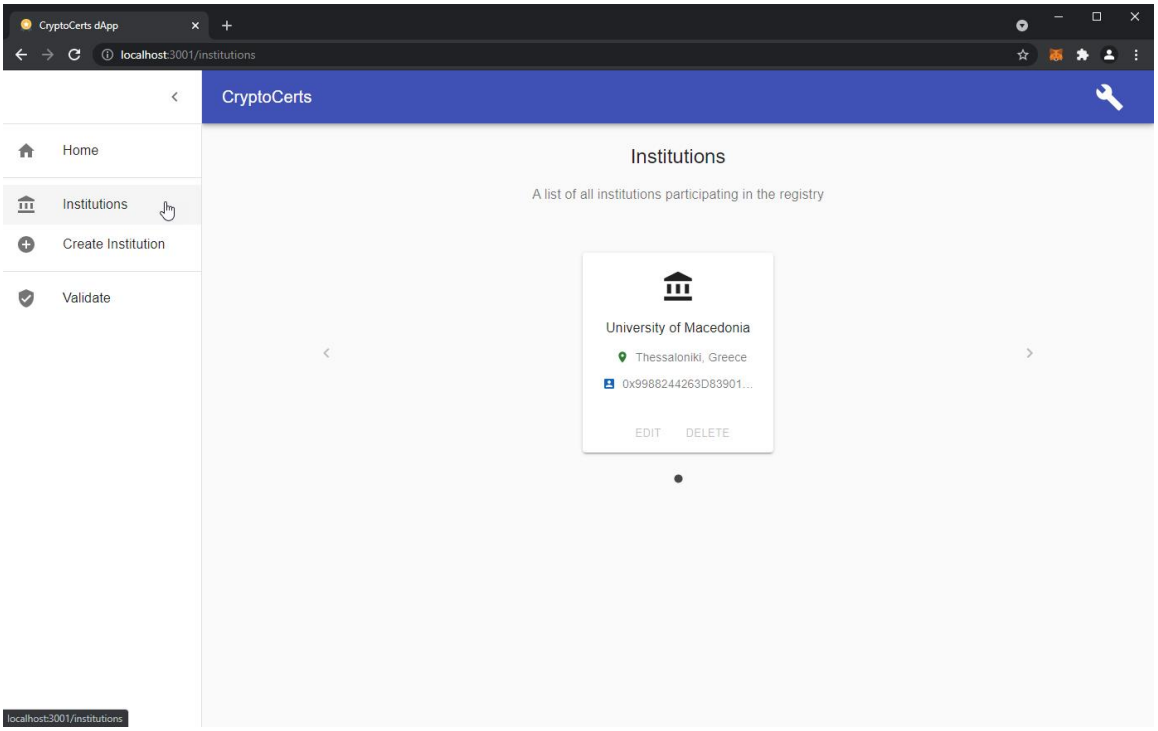
Screenshot 10 CryptoCerts waiting for an institution creation transaction to complete

A successful transaction through the Web3 provider will trigger the CryptoCerts contract, which will update its state according to the institution data provided and emit a new institution creation event. The CryptoCerts client, already a subscriber for these events, will capture it and update the user interface appropriately. The popup message gets discarded, and a relevant notification is displayed to the user, as we can see in Screenshot 11.



Screenshot 11 The new institution creation notification

To confirm that the new institution user has been registered, we navigate to the institution list page using the vertical navigation bar. As expected, our new university is now part of the blockchain, as depicted in Screenshot 12 below.



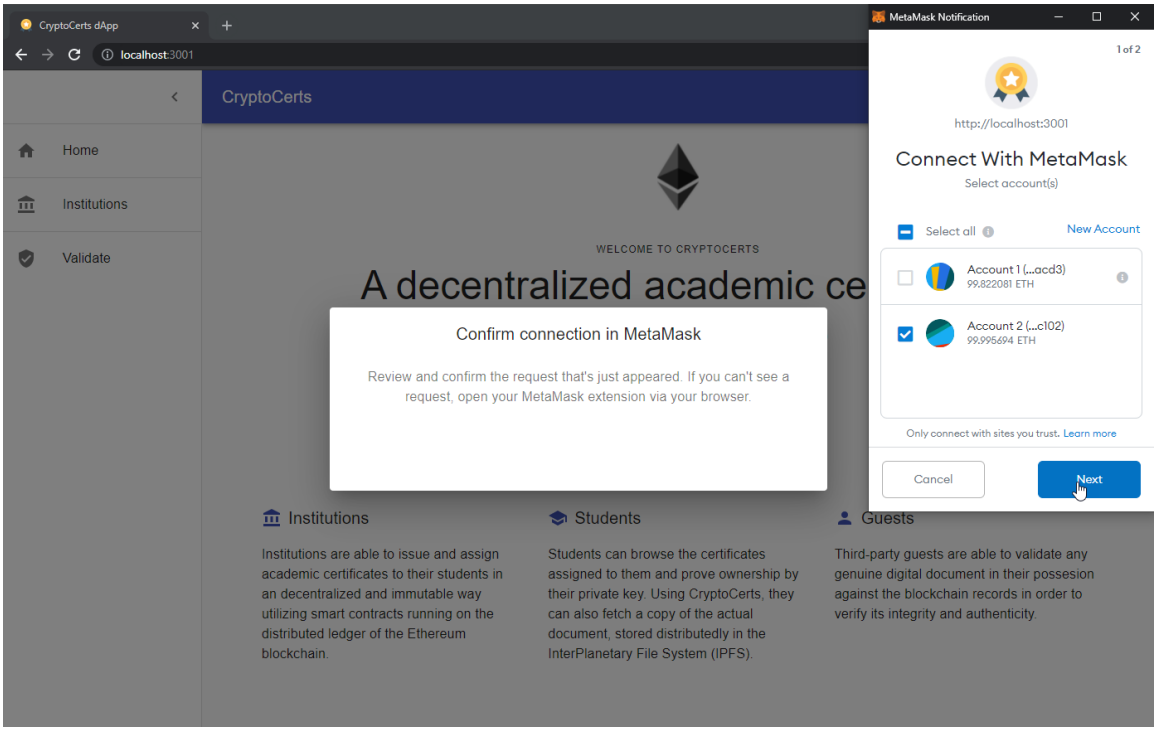
Screenshot 12 The institutions list populated

4.3.3 Issuing a certificate

Up to this point, we have presented how the interaction with the decentralized application is performed. The basic operations like connecting a user account and confirming transactions have been covered. Let us proceed to our primary business objective, creating a new academic certification document directly in the blockchain.

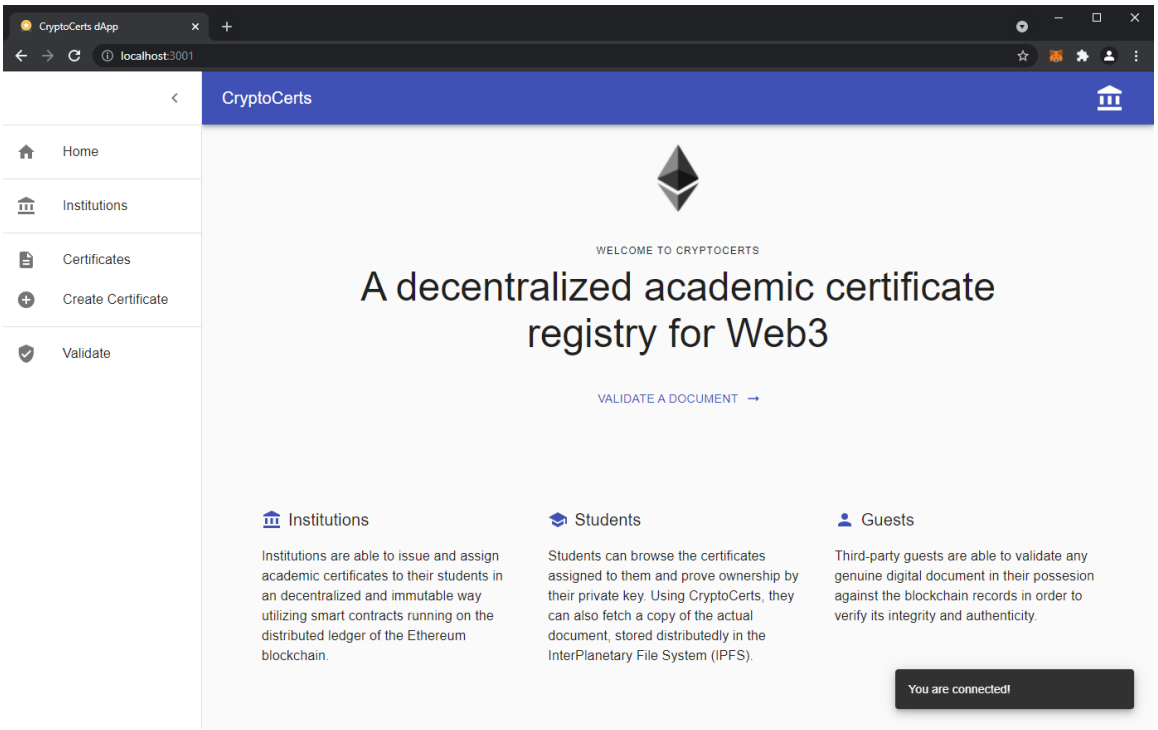
In the previous paragraph, acting as the application Administrator, the user created a new institution. We will use it to issue a new certificate to a student. To disconnect, we click on the Administrator roll icon and select “Disconnect.” A CryptoCert notification will confirm we are disconnected, and the application will redirect us to the landing page with only the guest areas available. We log out from our Web3 provider, and we will log back in with the keys that control our institution account.

As soon as we click the “Connect” button, the familiar connection windows will appear, as seen in Screenshot 13. This time we select our second account, which accounts for the institution we created earlier.



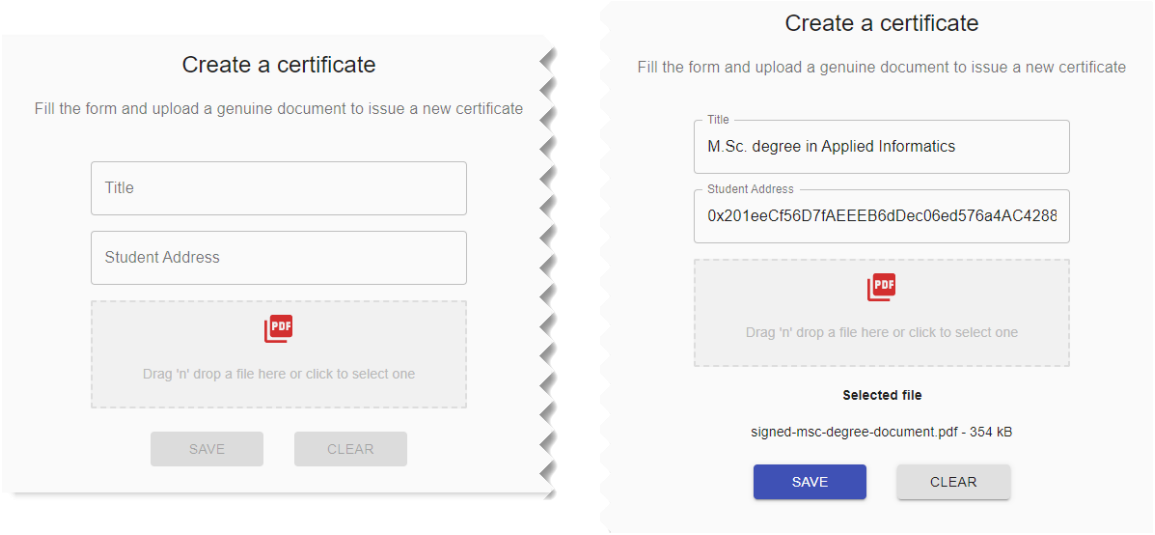
Screenshot 13 Connecting as an institution

Completing the operation will result in CryptoCerts identifying us as an institution. The user role icon appears in the upper right corner; a notification confirms we are connected, and the vertical menu renders some new items, “Certifications” and “Create Certificate.” All are seen in the following Screenshot 14.



Screenshot 14 The institution user screen

We click to create a new certificate, and we find ourselves transferred to a new form. We fill in the certificate title, the student address, and by clicking on the document drop area, we select the actual document containing an imaginary master's degree. If the drop area is inactive, we will have to ensure our IPFS daemon is running as this is a strict requirement to store the document. As this is not the case, we can see the form before and after the input in the following Screenshot 15.



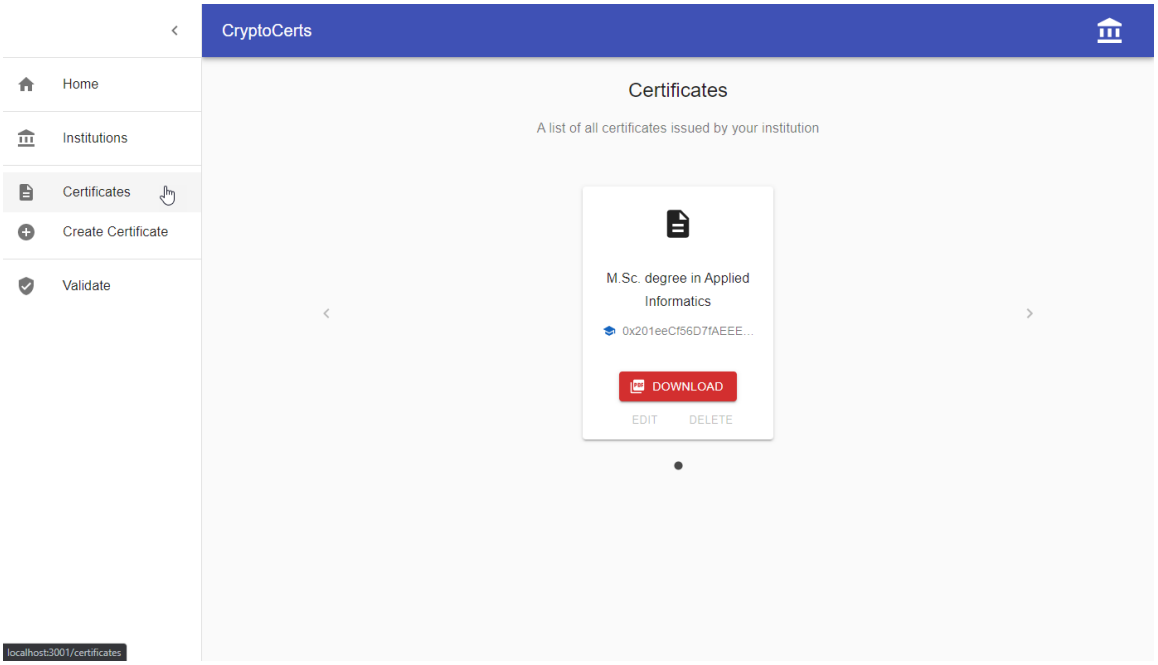
Screenshot 15 The certificate creation form before and after input

We are about to trigger another type of transaction against the smart contract certification creation interface this time. Clicking on the "Save" button will cause MetaMask and CryptoCerts to wait for our confirmation. As soon as it is done, the dapp client will upload the document to the IPFS network, which will create a new CID for it, basically the document hash. Similar to the institution creation transaction, the contract will update its state, storing the CID of the document under a new certificate record along with the student address. The student has been awarded a degree, and the blockchain is going to store it for eternity. A new notification appears, and the certificate is now available to be downloaded in the certificates list as depicted in [Screenshot 16].

We should note that the certificates list appearing here is filtered to include all the certificates issued by the connected institution. Switching to the student address, the user will face the certificates list populated with any certificates assigned to him by any institution in the network. We are going to skip this walkthrough as it is very similar to the steps already covered.

Last, we shall clarify that even if the certificates list is unavailable to the guests through the app as the current version, the blockchain records remain public. Anyone

knowing the CryptoCerts smart contract address could query its storage and extract all the data stored.

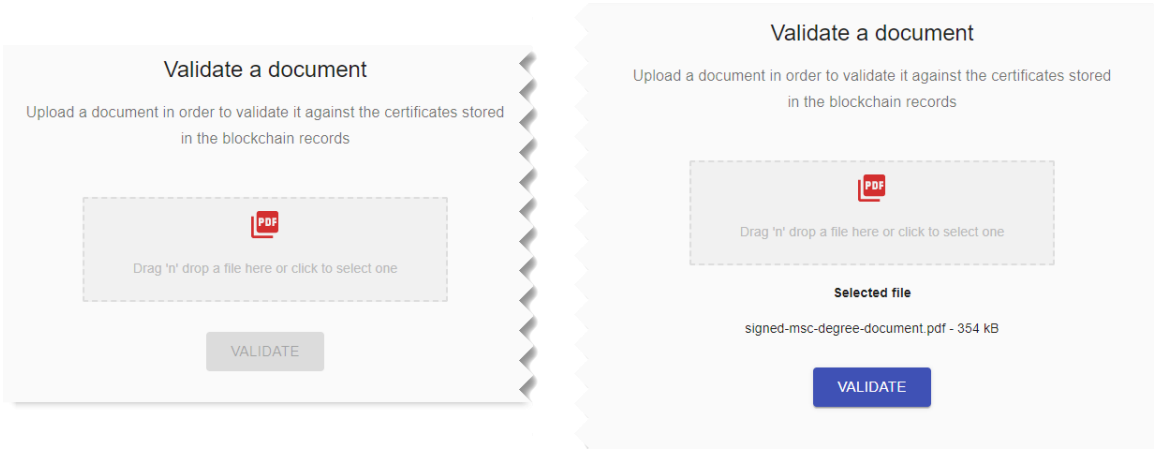


Screenshot 16 The certificates list populated

4.3.4 Validating a certificate

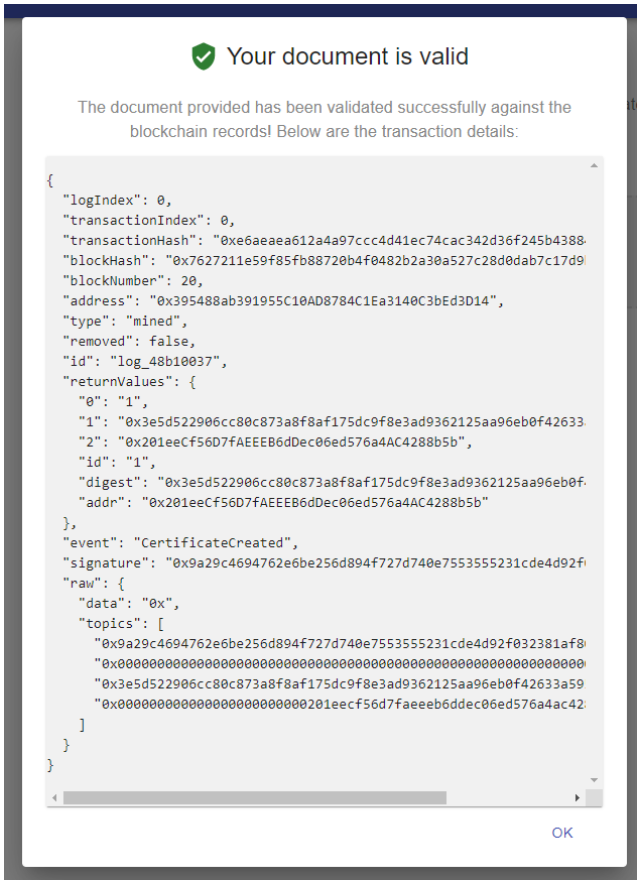
Last, we must impersonate a guest user trying to validate a certificate document against the decentralized registry. As mentioned before, this process does not require a connected account as it does not include any transactions.

Clicking either on the “Validate a document” button on the CryptoCerts landing page or the “Validate” menu item in the vertical navigation bar, the user navigates to the certificate validation form depicted in Screenshot 17.

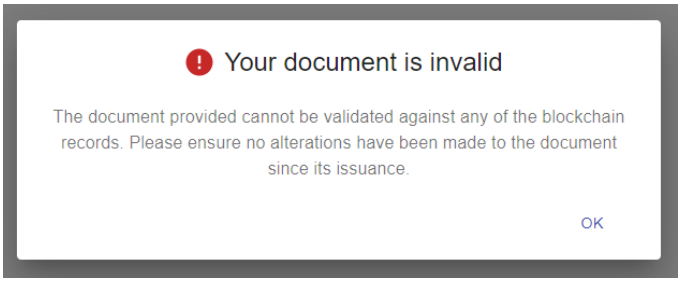


Screenshot 17 The certificate validation form before and after input

Similar to the certificate creation form, the form remains inactive if the IPFS daemon runs into any issues. In this case, the form accepts the certificate document we downloaded before as a student. When we press “Validate,” the client generates the hash of the selected document and searches for a match against the blockchain records. If the certificate document is registered and remains intact, the application will report a success message and the certificate creation transaction details in JSON format. This case can be seen in Screenshot 18. On the other hand, if the document supplied does not correspond to a certificate registered in the registry or its contents have been modified to a single bit, CryptoCerts will reject the file and print a relevant message as depicted in Screenshot 19.



Screenshot 18 The successful validation of a document



Screenshot 19 Invalid document response

4.4 System architecture

Following the architecture of the decentralized applications we reviewed in paragraph 2.4 , CryptoCerts consists of a web client for the frontend and a smart contract running on the blockchain for the backend part. In order to preserve true decentralization, no third-party services were utilized. The application is designed to run independently on every node connected to the blockchain network. The system architecture is depicted in Figure 15.

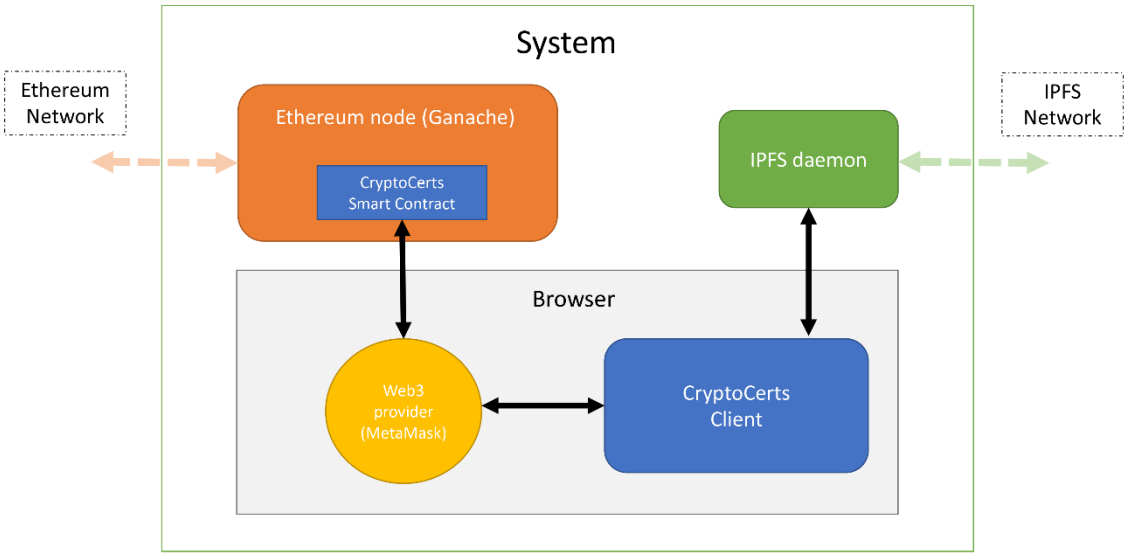


Figure 15 The CryptoCerts system architecture

The CryptoCerts client is written in JavaScript using the React framework, the Redux state manager, and the web3.js libraries. The backend contracts are written in Solidity utilizing the Truffle framework. Additionally, the IPFS network was utilized as decentralized documents hosting provider. An IPFS daemon should also operate in the system to support this functionality. Every part will be examined separately in the following paragraphs.

For the development version of CryptoCerts, the Ethereum node has been replaced by Ganache CLI, which acts as a blockchain emulator. For testing convenience, Ganache CLI is configured to generate a fixed number of accounts, crediting them with 100 ETH each in order for them to be able to transact with the blockchain. The exact accounts and unique private keys can be obtained from the Ganache CLI container output or the CryptoCerts contracts repository setup instructions. The preconfigured accounts can also be imported as a batch using a BIP32-compatible seed phrase to a compliant Web3 provider like MetaMask.

In order to improve the portability of the project, the environment has been containerized using Docker containers. The details of the Docker containers topology can be reviewed in Figure 16.

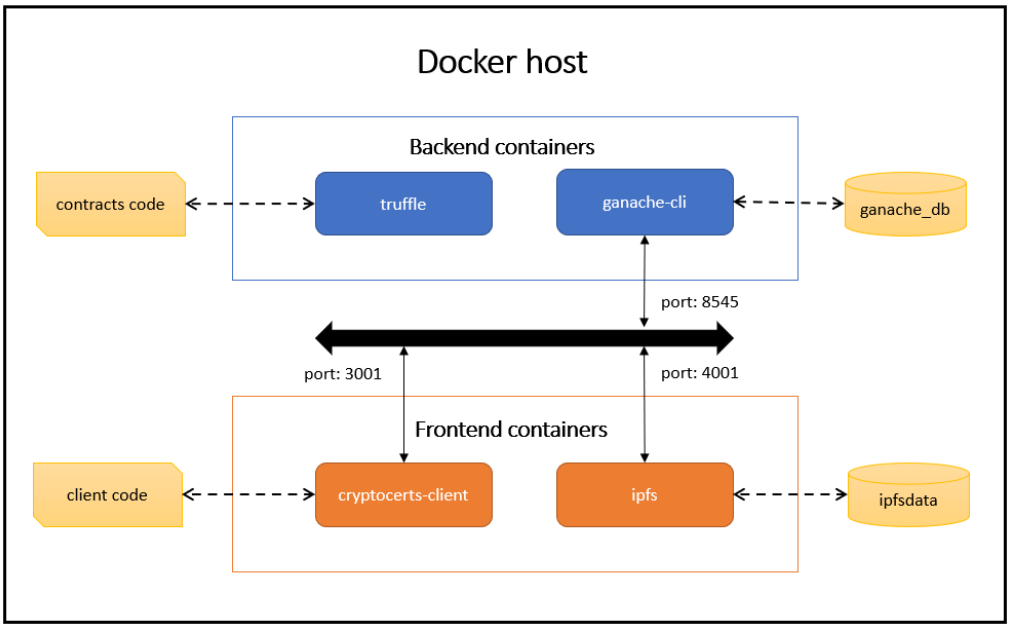


Figure 16 Docker stack topology

All data are stored in persistent Docker volumes allowing them to be wiped and start fresh at any time, in contrast to an immutable blockchain ledger or an actual IPFS node. All containers communicate through the host system, acting as a hub, with a set of port-forwarding rules in the Docker Compose manifests. At the same time, the IPFS node is configured to run in offline mode to reduce the network bandwidth overhead required for a development setup.

4.5 The Smart Contracts

CryptoCerts backend consists of the two contracts written in Solidity v0.7. Those are Migrations and CryptoCerts.

4.5.1 The Migrations contract

The Migrations contract is a helper provided by the Truffle framework for smart contract development. It provides the guidelines the framework needs to deploy a smart contract. As mentioned in paragraph 3.4 , Truffle migrations are compiled into JavaScript files that handle the evolution on the blockchain. The records of those migrations are written on the blockchain using the Migrations contract. We will describe how this task is performed by inspecting the contract code in Code 6.

```
1.  // SPDX-License-Identifier: MIT
2.  pragma solidity >=0.4.22 <0.8.0;
3.
4.  contract Migrations {
5.      address public owner = msg.sender;
6.      uint256 public last_completed_migration;
7.
8.      modifier restricted() {
9.          require(
10.             msg.sender == owner,
11.             "This function is restricted to the contract's owner"
12.         );
13.         _;
14.     }
15.
16.     function setCompleted(uint256 completed) public restricted {
17.         last_completed_migration = completed;
18.     }
19. }
20.
```

Code 6 Migrations.sol

The Migrations contract includes a popular pattern in Solidity as part of its `restricted` modifier. The modifier acts as an authorization middleware to the functions it is applied. Its simplest form validates that the contract owner performs the incoming call and rejects any calls that fail to meet that requirement. The owner's address is stored at the contract creation in the `owner` address variable.

As its main functionality, the Migration contract stores the integer corresponding to the last applied JavaScript migration script located in the migrations folder. This operation is done using the `setCompleted` function and the `last_completed_migration` variable.

4.5.2 The CryptoCerts contract

The CryptoCerts contract comprises the backend business logic of the CryptoCerts decentralized application. This contract is where the institutions and the certificates are registered as part of the blockchain state.

As mentioned in subsection 3.2 , smart contracts could be parallelized with classes in object-oriented programming, and as such, they can use design patterns like inheritance. Inheritance can be used for logical inheritance, but it can also be used simply to organize the code by grouping similar logic into different contracts. The CryptoCerts contract consists of five contracts: Context, Ownable, InstitutionFactory, CertificateFactory, and the CryptoCerts wrapper contract. An atypical UML class diagram describing the relations between the contracts can be found in Figure 17.

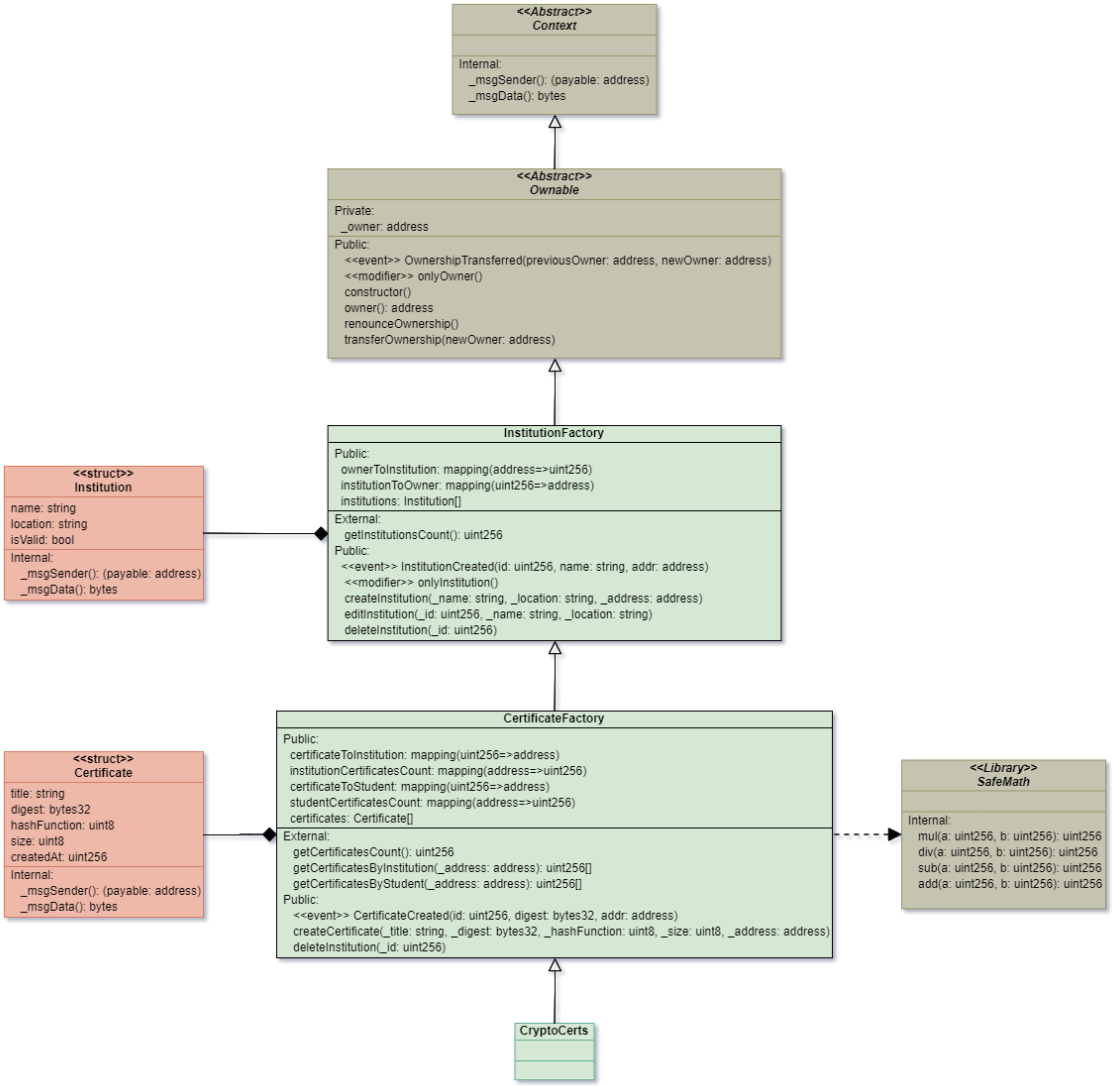


Figure 17 CryptoCerts UML diagram

- ***Context and Ownable***

Context and Ownable contracts are part of the OpenZeppelin package described in paragraph 3.3 . The first provides helper functions our main contracts utilize for the current execution context, including the transaction sender and its data. The latter offers a basic access control mechanism, where there is an account named as the owner that can be granted exclusive access to specific functions. This module is a more polished version of the `restricted` modifier, the Migrations contract included. Both Context and Ownable contracts are defined as abstract, and they provide their functionality through inheritance.

- ***SafeMath***

The SafeMath contract is a library. Libraries in the context of Solidity are reusable contracts offering functions other contracts can call. SafeMath is a wrapper over Solidity's arithmetic operations with added overflow checks. Arithmetic operations as of Solidity v0.7 wrap on overflow. This behavior can easily result in bugs because programmers usually assume that an overflow raises an error, which is standard in high-level programming languages. SafeMath restores this expectation by reverting the transaction when an arithmetic operation overflows. Solidity v0.8 announced that it provides this functionality natively. However, for this project's scope but we should rely on an external library.

- ***InstitutionFactory***

InstitutionFactory is the concrete parent of CryptoCerts backend implementation. As the name implies, this contract implements a factory pattern for creating institutions and hosts all the relevant CRUD operations. The Solidity implementation can be reviewed in Code 7.

```
1.  // SPDX-License-Identifier: UNLICENSED
2.  pragma solidity ^0.7.0;
3.
4.  import "../node_modules/@openzeppelin/contracts/access/Ownable.sol";
5.
6.  contract InstitutionFactory is Ownable {
7.      event InstitutionCreated(
8.          uint256 indexed id,
9.          string name,
10.         address indexed addr
11.     );
12.
13.     struct Institution {
14.         string name;
15.         string location;
16.         bool isValid;
17.     }
18.
19.     mapping(address => uint256) public ownerToInstitution;
20.     mapping(uint256 => address) public institutionToOwner;
21.
22.     Institution[] public institutions;
23.
24.     /**
25.      * @dev Throws if called by any account who doesn't beloong to an Institution.
26.      */
27.     modifier onlyInstitution() {
28.         require(
29.             ownerToInstitution[_msgSender()] != 0,
30.             "Caller is not an institution owner"
31.         );
32.         _;
33.     }
34.
```

```
35.     function createInstitution(string memory _name, string memory _location, address
    _address) public onlyOwner {
36.         institutions.push(Institution(_name, _location, true));
37.
38.         uint256 id = institutions.length;
39.         ownerToInstitution[_address] = id;
40.         institutionToOwner[id] = _address;
41.
42.         InstitutionCreated(id, _name, _address);
43.     }
44.
45.     function editInstitution(uint256 _id, string memory _name, string memory
    _location) public onlyOwner {
46.         institutions[_id].name = _name;
47.         institutions[_id].location = _location;
48.     }
49.
50.     function deleteInstitution(uint256 _id) public onlyOwner {
51.         Institution memory institution = institutions[_id];
52.         institution.isValid = false;
53.         ownerToInstitution[institutionToOwner[_id]] = 0;
54.         institutionToOwner[_id] = address(0);
55.     }
56.
57.     function getInstitutionsCount() external view returns (uint256) {
58.         return institutions.length;
59.     }
60. }
61.
```

Code 7 InstitutionFactory.sol

An institution is represented by a struct object containing its name and location and a validity flag. The institution objects are pushed into a storage variable array named `institutions`, and its index is being used the institution identifier it hosts. This collection is essentially eternally written on the blockchain. We do not plan to reorder the array, as this will be too costly in gas, rendering our contract unusable. So, keep the index as an institution ID is considered safe. We need quick lookups based on the institution ID and the owning address, and we implemented them as mappings. The `ownerToInstitution` mapping links an account address to an institution ID, while the `institutionToOwner` handles the reverse operation. Last, the InstitutionFactory contract defines a new type of event called `InstitutionCreated`. This event will be used for updating our client interface. It contains the institution identifier, its name, and its account address.

The interface of the contract consists of the functions `createInstitution`, `editInstitution`, `deleteInstitution` and `getInstitutionsCount`. Their operations are pretty obvious. `createInstitution` creates a new institution object and stores it in the `institutions` storage array. It populates our lookup mappings and emits an event that a new institution has been created. We should note that this function utilizes the `onlyOwner`

access modifier from the Ownable parent contract, allowing its execution only to the contract owner, thus the application Administrator. The `editInstitution` and `deleteInstitution` functions update and void an institution in the storage array. `getInstitutionsCount` is a helper returning the size of the storage array, and it is marked as a view function, meaning that its execution does not require any gas.

- ***CertificateFactory***

The CertificateFactory contract is the heart of our certificate registry. As expected, it handles the certification creation and responds to any read-only queries regarding the resource. We will review its implementation as seen in Code 9.

```
1.  // SPDX-License-Identifier: UNLICENSED
2.  pragma solidity ^0.7.0;
3.
4.  import "../node_modules/@openzeppelin/contracts/math/SafeMath.sol";
5.  import "./InstitutionFactory.sol";
6.
7.  contract CertificateFactory is InstitutionFactory {
8.      using SafeMath for uint256;
9.
10.     event CertificateCreated(
11.         uint256 indexed id,
12.         bytes32 indexed digest,
13.         address indexed addr
14.     );
15.
16.     struct Certificate {
17.         string title;
18.         bytes32 digest;
19.         uint8 hashFunction;
20.         uint8 size;
21.         uint256 createdAt;
22.     }
23.
24.     mapping(uint256 => address) public certificateToInstitution;
25.     mapping(address => uint256) public institutionCertificatesCount;
26.
27.     mapping(uint256 => address) public certificateToStudent;
28.     mapping(address => uint256) public studentCertificatesCount;
29.
30.     Certificate[] public certificates;
31.
32.     function createCertificate(
33.         string memory _title,
34.         bytes32 _digest,
35.         uint8 _hashFunction,
36.         uint8 _size,
37.         address _address
38.     ) public {
39.         Certificate memory cert = Certificate(
40.             _title,
41.             _digest,
42.             _hashFunction,
43.             _size,
44.             block.timestamp
45.         );
46.         certificates.push(cert);
47.         uint256 id = certificates.length;
```

```
48.
49.         certificateToInstitution[id] = msg.sender;
50.         institutionCertificatesCount[msg.sender] =
    institutionCertificatesCount[msg.sender].add(1);
51.
52.         certificateToStudent[id] = _address;
53.         studentCertificatesCount[_address] =
    studentCertificatesCount[_address].add(1);
54.
55.         CertificateCreated(id, _digest, _address);
56.     }
57.
58.     function getCertificatesCount() external view returns (uint256) {
59.         return certificates.length;
60.     }
61.
62.     function getCertificatesByInstitution(address _address) external view returns
    (uint256[] memory) {
63.         uint256[] memory result = new
    uint256[](institutionCertificatesCount[_address]);
64.
65.         uint256 counter = 0;
66.         for (uint256 i = 0; i < certificates.length; i++) {
67.             if (certificateToInstitution[i] == _address) {
68.                 result[counter] = i;
69.                 counter++;
70.             }
71.         }
72.
73.         return result;
74.     }
75.
76.     function getCertificatesByStudent(address _address) external view returns
    (uint256[] memory) {
77.         uint256[] memory result = new uint256[](studentCertificatesCount[_address]);
78.
79.         uint256 counter = 0;
80.         for (uint256 i = 0; i < certificates.length; i++) {
81.             if (certificateToStudent[i] == _address) {
82.                 result[counter] = i;
83.                 counter++;
84.             }
85.         }
86.
87.         return result;
88.     }
89. }
90.
```

Code 8 CertificateFactory.sol

A struct object represents a certificate consisted of its title, its document hash, the hash function code, the hash size, and finally, its creation time as the block number it was created. To keep our registry futureproof, we store document hashes based on the Multihash protocol [67]. So in case, IPFS changes the hash function or the length of its CIDs, CryptoCerts will still be compatible. The certificate struct also takes advantage of the Solidity struct packing feature by placing smaller integer sub-types next to each other in the struct. In our case, `_hashFunction` uint8 and `_size` uint8 properties will be clustered into a uint16, saving valuable gas from storage.

Similar to its parent, CertificateFactory utilizes a storage array for keeping the certificate objects. We need to filter the `certificates` array by institution and student to feed the corresponding client lists. Storing them into separate arrays is strictly prohibited in the context of smart contract development. The excessive storage required to maintain them will skyrocket the gas cost. Instead, we are going to rely on rebuilding these arrays in memory on the fly. We need a certificate id to address mapping and an address to certificate count mapping for this approach. We replicate this structure per associated filter, resulting in the `certificateToInstitution`, `institutionCertificatesCount`, `certificateToStudent`, and `studentCertificatesCount` mappings. CertificateFactory also defines an event for newly created certificates containing their ID, document hash, and student address. The client uses this event log to update the user interface and validate documents against the blockchain records.

The CertificateFactory contract interface includes the `createCertificate`, `getCertificatesByInstitution`, `getCertificatesByStudent` and `getCertificatesCount` functions. Certificate creation forms a new certificate object based on the input data received. The object is being pushed into the certificates storage array receiving its identifier from the array index. Lookup mappings and counts are being updated. Incrementing a count here is performed using the overflow-safe arithmetic operations of the SafeMath library. Last, a new `CertificateCreated` event is being fired. `getCertificatesByInstitution` and `getCertificatesByStudent` are quite similar in implementing the certificates filtering based on the address passed. Last, `getCertificatesCount` is a helper function returning the total count of the certificates in the registry. All three getters are defined as views marking them free in terms of gas cost.

- ***CryptoCerts***

This contract stands last in the line of inheritance, and it essentially renames and wraps the backend functionality under a single contract. Its boilerplate code can be reviewed in Code 9.

```
1.  // SPDX-License-Identifier: UNLICENSED
2.  pragma solidity ^0.7.0;
3.
4.  import "./CertificateFactory.sol";
5.
6.  contract CryptoCerts is CertificateFactory {
7.      //
8.  }
9.
```

Code 9 CryptoCerts.sol

4.6 The Client

The CryptoCerts client is a complete web application built using modern technologies rather than a single web page. It is responsible for rendering the CryptoCerts interface in a web browser, allowing the user to interact with the smart contracts securely and transparently.

The web client is written in JavaScript ES6, also known as ECMAScript 2015, the second major revision of JavaScript. The application is composed of several web components following the approach of the React web framework. The modular architecture allows a clear separation of concerns and enhances scalability.

For the project bootstrapping, we used the officially-supported Create React Application (CRA) setup script. CRA provides a straightforward way to maintain a scalable directory structure and a set of utility scripts to manage dependencies and spin up a development server. The source directory structure of the CryptoCerts client is depicted in Figure 18.

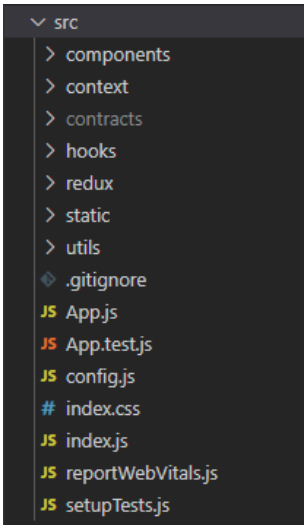


Figure 18 CryptoCerts client source directory structure

Truffle populates the `contracts` directory with a series of JSON files containing the smart contract Application Binary Interfaces (ABIs). The ABIs expose the supported contract interface for the web3.js library to communicate successfully. Any new deployment to the blockchain will update these files to match the version of the contract on-chain.

In the following paragraphs, we will describe briefly every component and state controller comprising the CryptoCerts React client.

4.6.1 Components

The React application consists of several reusable components. CryptoCerts is written entirely using React Functional Components, which are essentially JavasScript functions that return JSX. The implementation details of each component are considered out of scope for the current analysis. However, we can briefly group them into the following three categories based on their role and functionality:

- **Layout** – Components that contribute to the application layout
  - **MainLayout**: Dictates the application layout
  - **TopBar**: Displays the top navigation bar
  - **Header**: Displays a consistent header on each page
  - **Content**: Wraps the main content area
  - **SliderDrawer**: Displays the vertical collapsible navigation menu
  - **ProtectedRoute**: Renders a menu item using the React Router
- **Pages** – Components that participate in particular pages rendering
  - **Home**: Renders the CryptoCerts landing page
  - **CertificateCard**: Renders a card representing a certificate
  - **CertificateForm**: Renders the certificate creation form
  - **CertificateList**: Displays the certificates list page
  - **DocumentDropzone**: Renders the document dropzone area
  - **InstitutionCard**: Renders a card representing an institution
  - **InstitutionForm**: Renders the institution creation form
  - **InstitutionList**: Displays the certificates list page
  - **MaterialCarousel**: Renders a carousel containing a list of items
  - **ValidatePage**: Displays the document validation page
- **Dialogs** – Informative or warning dialogs notifying the user about an action or an event
  - **Web3AlertDialog**: Displays an alert dialog about browser and Web3 provider incompatibility
  - **MetamaskDialog**: Renders a loading dialog when an action is required on MetaMask
  - **NetworkAlertDialog**: Displays an alert dialog about the connection to a wrong network

- **ValidDocumentDialog**: Renders the results dialog for a valid document
- **InvalidDocumentDialog**: Renders the results dialog for an invalid document
- **Notification**: Displays a notification in the bottom right corner
- **Spinner**: Renders the spinning loader on a MetamaskDialog

4.6.2 Context

CryptoCerts utilizes context to pass data into several components in a parent-to-child or top-down fashion. The following context providers were implemented:

- **ConnectionProvider**: Handles the connection state with the Web3 provider and listens to the institution and certificate creation events.
- **DrawerProvider**: Handles the state of the vertical collapsible navigation bar
- **NotificationProvider**: Handles the state of on-screen notifications

4.6.3 Hooks

The web client is written in React v17; thus, it supports the new concept of hooks. CryptoCerts client uses hooks to provide universal access to the global state without relying on components hierarchy. The following hooks have been implemented:

- **useCryptoCerts**: After the initial connection with the Web3 provider, this hook provides the interface to the CryptoCerts smart contract
- **useInterval**: Utility hook which provides a ticker for the application to poll the blockchain state
- **useIpfs**: Handles the connection with the IPFS daemon exposing an interface that supports document upload/download and CID generation

4.6.4 Redux store

CryptoCerts uses React with Redux to maintain a state container for all its features to synchronize and function correctly.

One of the states a decentralized application has to be aware of is the state of the blockchain. As part of a network, the contract state can change with any block added to the chain. For this operation, the Truffle suite provides a preconfigured Redux store with the name Drizzle that binds to a contract interface and notifies the application of any

changes. Unfortunately, at the time of this writing, Drizzle is quite experimental with minimal documentation. As a result, a custom implementation using Redux was promoted for CryptoCerts.

The following slices were introduced as part of our Redux store:

- **connectionSlice**: Holds the state of the connection with the Web3 provider. This state includes the active account and user role to enable or disable particular features.
- **institutionsSlice**: Holds the contract state dedicated to institutions. The supported operations include the creation of a new entity and fetching an institution or a list.
- **institutionsSlice**: Holds the contract state dedicated to institutions. The supported operations include the creation of a new entity and fetching an institution or a list.
- **certificatesSlice**: Holds the contract state dedicated to certificates. The supported operations include the creation of a new entity and fetching a certificate or a list based on the active user account.

5 Conclusion

Decentralized applications are still a very new concept. The majority of people and businesses are unaware of the tangible benefits of blockchain that go far beyond the initial use case of cryptocurrencies. The global community is already experimenting with the technology, but using it still requires a leap of faith. At the same time, foremost industry leaders like Microsoft and Facebook rush to secure a spot in the blockchain space.

5.1 Overview

This thesis presented the need for a decentralized web which may become more apparent in the following years. We were able to identify the key concepts that enable this transformation and describe an evolving ecosystem around the second generation of blockchain platforms. The decentralized application development is not still a blueprint but an emerging new type of software engineering.

The second part of this dissertation proves our case with the implementation of a decentralized application that mitigates a real-world problem like academic certificate forgery. We removed any intermediaries and provided an open platform that preserves academic credentials transparency directly from the issuing authority. We complement our implementation with the integration of IPFS as decentralized storage.

We successfully relied on user keys for their identification and blockchain interaction. Additionally, we showcased that an inclusive user experience can be achieved by focusing on the users' perspective. We also managed to do so without the need for third-party services while preserving our approach's decentralization.

From the software engineering perspective, decentralized application development proved to have many similarities to conventional software development. This assertion is not only based on the fact that dapps share typical web clients with web applications. The same agile software methodologies were able to be applied as described from analysis to verification.

Last, particular traits of smart contract development were pointed out through the implementation. The similarity of contracts to classes allows the adoption of object-oriented programming patterns like inheritance. The aspect of the gas cost was highlighted as a crucial concern that defined every step of our code design.

5.2 Future Work

The development of decentralized applications proved to be too extensive to be covered through a single dissertation. As part of future work, we could identify various additions and enhancements to the current implementation.

An exciting topic to focus on is smart contract upgradeability. Smart contracts in Ethereum are immutable by default. Once created, there is no way to be altered. However, there are some cases when it is desirable to be able to modify them. We could, for example, implement some additional features like a certificate update in CryptoCerts. For that purpose, we could utilize proxy contracts. A proxy contract essentially acts like a wrapper the user interacts with, and it is responsible for forwarding transactions to and from the implementation contract.

Other concerns around decentralized applications include user identity issues or unlinkability. It would be interesting to investigate how a student could recover the certificates assigned to them in case of a lost key. An intriguing scenario could also cover the ability of a student to remain unlinked to their certificates, only to provide proof for some of them or to specific third parties. There are fascinating developments in the domain of zero-knowledge proofs, like zk-SNARKs, that could assist in implementing such functionalities.

6 References

[1 Investopedia, "Dotcom Bubble," 25 June 2019. [Online]. Available: https://www.investopedia.com/terms/d/dotcom-bubble.asp. [Accessed 30 December 2020].

[2 T. O'Reilly, "What Is Web 2.0," 9 September 2005. [Online]. [Accessed 30 December 2020].

[3 International Telecommunication Union (ITU), "The World in 2014 - ICT Facts and Figures," April 2014. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf. [Accessed 31 December 2020].

[4 N. Yau, "Rise of the Data Scientist," 4 June 2009. [Online]. Available: https://flowingdata.com/2009/06/04/rise-of-the-data-scientist/. [Accessed 31 12 2020].

[5 DataReportal, "Global Digital Growth October 2020," [Online]. Available: https://datareportal.com/reports/digital-2020-october-global-statshot. [Accessed 30 December 2020].

[6 Apple Inc., "Consolidated Financial Statements - Q3 2018," 31 July 2018. [Online]. Available: https://s2.q4cdn.com/470004039/files/doc_financials/2018/q3/Q3FY18ConsolidatedFinancialStatements.pdf. [Accessed 01 January 2021].

[7 J. Cramer, "Does Your Portfolio Have FANGs?," 5 Feb 2013. [Online]. Available: https://www.cnbc.com/id/100436754. [Accessed 1 January 2021].

[8 Financial Times, "Rise of the US mega-caps creates shaky ‘top-heavy’ market," 22 July 2020. [Online]. Available: https://www.ft.com/content/95aeb21d-8ade-48f8-82e7-cbf4b85657aa. [Accessed 1 January 2021].

[9 Amazon Web Services, "Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region," 25 November 2020. [Online]. Available: https://aws.amazon.com/message/11201/. [Accessed 1 January 2021].

[1 Google Cloud, "Google Cloud Infrastructure Components Incident #20013," 14 December 2020. [Online]. Available: https://status.cloud.google.com/incident/zall/20013. [Accessed 1 January

2021].

[1 Equifax Inc., "Equifax Announces Cybersecurity Incident Involving Consumer Information," 7 September 2017. [Online]. Available: https://www.equifaxsecurity2017.com/2017/09/07/equifax-announces-cybersecurity-incident-involving-consumer-information/. [Accessed 1 January 2021].

[1 The New York Times Company, "Ex-Worker at C.I.A. Says He Leaked Data on Surveillance," 9 June 2013. [Online]. Available: https://www.nytimes.com/2013/06/10/us/former-cia-worker-says-he-leaked-surveillance-data.html. [Accessed 1 January 2021].

[1 The Guardian, "Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach," 17 March 2018. [Online]. Available: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election. [Accessed 02 January 2021].

[1 B. Sterling, Interviewee, *Bruce Sterling on Why It Stopped Making Sense to Talk About 'The Internet' in 2012*. [Interview]. 27 December 2012.

[1 S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 31 October 2018. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 2 January 2021].

[1 R. Rivest, T. H. Cormen and C. Leiserson, Introduction to Algorithms, Massachusetts: The MIT Press, 1990.

[1 F. L. Bauer, Decrypted Secrets - Methods and Maxims of Cryptology, Berlin: Springer, 1997.

[1 National Bureau of Standards (NBS), "Announcing the Data Encryption Standard (DES)," 15 January 1977. [Online]. Available: https://csrc.nist.gov/CSRC/media/Publications/fips/46/archive/1977-01-15/documents/NBS.FIPS.46.pdf. [Accessed 07 09 2021].

[1 National Institute of Standards and Technology (NIST), "Announcing the Advanced Encryption Standard (AES)," 26 November 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. [Accessed 7 September 2021].

[2 W. Dieffie and M. Hellman, "New Directions in Cryptography," 6 November 1976. [Online]. Available: https://ee.stanford.edu/~hellman/publications/24.pdf. [Accessed 7 September 2021].

[2 R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital," 1 February 1978. [Online]. Available: https://people.csail.mit.edu/rivest/Rsapaper.pdf. [Accessed 7 September 2021].

[2 Software Engineering Institute, “MD5 vulnerable to collision attacks,” 31 December 2008. [Online]. Available: https://www.kb.cert.org/vuls/id/836068. [Accessed 7 September 2021].

[2 National Institute of Standards (NIST), "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," 4 August 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. [Accessed 8 September 2021].

[2 S. S. Haber, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, no. 2, p. 99–111, January 1991.

[2 N. Szabo, "Bit Gold," 29 December 2005. [Online]. Available: https://nakamotoinstitute.org/bit-gold/. [Accessed 06 October 2021].

[2 X. Li, P. Jiang, T. Chen, T. Chen and Q. Wen, "A Survey on the Security of Blockchain Systems," *Future Generation Computer Systems,* August 2017.

[2 Z. Sheping, Y. Yuanyuan, L. Jing, Q. Cheng and Z. Jiangming, "Research on the Application of Cryptography on the Blockchain," *Journal of Physics: Conference Series,* vol. 1168, no. 3, 2019.

[2 Y. Sompolinsky and A. Zohar, "Bitcoin's Underlying Incentives - The unseen economic forces that govern the Bitcoin protocol," *ACM Queue,* vol. 15, no. 5, September 2017.

[2 A. Shapiro, "Blockchains: what are they and how do they work?," January 2019. [Online]. Available: https://www.researchgate.net/publication/348150425_Blockchains_what_are_t

hey_and_how_do_they_work. [Accessed 06 October 2021].

[3 R. Zhang, *Analyzing and Improving Proof-of-Work Consensus Protocols,* Arenberg Doctoral School - Faculty of Engineering Science, 2019.

[3 E. Muzzy, "What Is Proof of Stake?," ConsenSys Inc., 15 May 2020. [Online]. Available: https://consensys.net/blog/blockchain-explained/what-is-proof-of-stake/. [Accessed 06 October 2021].

[3 V. Buterin, "On Public and Private Blockchains," The Ethereum Foundation, 07 August 2015. [Online]. Available: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/. [Accessed 06 October 2021].

[3 N. Szabo, "Smart Contracts," 1994. [Online]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html. [Accessed 06 October 2021].

[3 L. W. Cong and Z. He, "Blockchain disruption and smart contracts," March 2018. [Online]. Available: https://www.nber.org/system/files/working_papers/w24399/w24399.pdf. [Accessed 06 October 2021].

[3 J. Stark, "Making Sense of Blockchain Smart Contracts," CoinDesk, 04 June 2016. [Online]. Available: https://www.coindesk.com/markets/2016/06/04/making-sense-of-blockchain-smart-contracts/. [Accessed 06 October 2021].

[3 The Ethereum Foundation, "Introduction to dapps," [Online]. Available: https://ethereum.org/el/developers/docs/dapps/. [Accessed 06 October 2021].

[3 M. Swan, Blockchain, O'Reilly Media, Inc., 2015.

[3 V. Buterin, "A next-generation smart contract and decentralized application platform," 14 January 2014. [Online]. Available: https://ethereum.org/en/whitepaper/. [Accessed 04 October 2021].

[3 W. Chen, T. Zhang, Z. Chen, Z. Zheng and Y. Lu, "Traveling the token world: A

graph analysis of Ethereum ERC20 token ecosystem," April 2020. [Online]. Available: https://www.researchgate.net/publication/341126900_Traveling_the_token_world_A_graph_analysis_of_Ethereum_ERC20_token_ecosystem. [Accessed 06 October 2021].

[4 L. Lesavre, P. Varin and D. Yaga, "Blockchain Networks: Token Design and Management Overview," National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2021.

[4 F. Vogelsteller and V. Buterin, "EIP-20: Token Standard," November 2015. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20. [Accessed 06 October 2021].

[4 E. William, S. Dieter, J. Evans and N. Sachs, "EIP-721: Non-Fungible Token Standard," January 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721. [Accessed 06 October 2021].

[4 Ethereum Foundation, "Solidity documentation," Ethereum Foundation, 13 July 2020. [Online]. Available: https://docs.soliditylang.org/en/v0.7.0/index.html. [Accessed 04 October 2021].

[4 The Linux Foundation, "SPDX IDs," The Software Package Data Exchange (SPDX), [Online]. Available: https://spdx.dev/ids/. [Accessed 06 October 2021].

[4 R. Infante, Building Ethereum Dapps - Decentralized applications on the Ethereum blockchain, Shelter Island, New York, USA: Manning, 2019.

[4 L. Hollander, "Understanding event logs on the Ethereum blockchain," March 2020. [Online]. Available: https://medium.com/mycrypto/understanding-event-logs-on-the-ethereum-blockchain-f4ae7ba50378. [Accessed 06 October 2021].

[4 OpenZeppelin, "OpenZeppelin," [Online]. Available: https://openzeppelin.com. [Accessed 06 October 2021].

[4 OpenZeppelin, "Contracts - OpenZeppelin Docs," OpenZeppelin, [Online]. Available:

https://docs.openzeppelin.com/contracts/4.x/. [Accessed 06 October 2021].

[4 ConsenSys Software Inc., "Truffle - Overview - Documentation," [Online]. Available: https://www.trufflesuite.com/docs/truffle/overview. [Accessed 06 October 2021].

[5 J. Chittoda, Mastering Blockchain Programming with Solidity, Birmingham, UK: Packt Publishing, 2019.

[5 ConsenSys Software Inc., "Ganache - Overview - Documentation," [Online]. Available: https://www.trufflesuite.com/docs/ganache/overview. [Accessed 06 October 2021].

[5 Protocol Labs, "IPFS Documentation," [Online]. Available: https://docs.ipfs.io/. [Accessed 06 October 2021].

[5 Protocol Labs, "IPLD Documentation," [Online]. Available: https://ipld.io/docs/. [Accessed 06 October 2021].

[5 C. Helbling, "Directed Graph Hashing," 16 February 2020. [Online]. Available: https://arxiv.org/pdf/2002.06653.pdf. [Accessed 06 October 2021].

[5 S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, "A Scalable Content-Addressable Network," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, p. 161–172, October 2001.

[5 Facebook Inc., "React - A JavaScript library for building user interfaces," Facebook Inc., [Online]. Available: https://reactjs.org/. [Accessed 06 October 2021].

[5 RisingStack Engineering, "The History of React.js on a Timeline," 21 September 2021. [Online]. Available: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/. [Accessed 06 October 2021].

[5 D. Abramov, "Redux: A Predictable State Container for JS Apps," [Online]. Available: https://redux.js.org/. [Accessed 06 October 2021].

[5 ChainSafe Systems, "web3.js - Ethereum JavaScript API," [Online]. Available: https://web3js.readthedocs.io/. [Accessed 06 October 2021].

[6 MetaMask, "MetaMask - A crypto wallet & gateway to blockchain apps," ConsenSys Inc., [Online]. Available: https://metamask.io/. [Accessed 06 October 2021].

[6 Docker, Inc., "Docker - Empowering App Development for Developers," [Online]. Available: https://www.docker.com/. [Accessed 06 October 2021].

[6 R. Osnat, "A Brief History of Containers: From the 1970s Till Now," 10 January 2020. [Online]. Available: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016. [Accessed 06 October 2021].

[6 Docker, Inc., "Docker Documentation - Overview of Docker Compose," [Online]. Available: https://docs.docker.com/compose/. [Accessed 06 October 2021].

[6 Academic Credentials Evaluation Institute, Inc. (ACEI),, «Diploma Mills & Fake Degrees: A billion $$$ industry,» 24 May 2019. [Ηλεκτρονικό]. Available: https://acei-global.org/diploma-mills-fake-degrees-a-billion-industry/. [Πρόσβαση 06 October 2021].

[6 J. Saidi-Kuehnert, «Fighting Diploma Fraud & Protecting Credential Integrity with Technology,» Academic Credentials Evaluation Institute (ACEI), 08 February 2019. [Ηλεκτρονικό]. Available: https://acei-global.blog/2019/02/08/fighting-diploma-fraud-protecting-credential-integrity-with-technology/. [Πρόσβαση 06 October 2021].

[6 ConsenSys Inc., "Rimble Guides - Dapp patterns," [Online]. Available: https://rimble.consensys.design/guides/ux/connect-a-wallet-conditions. [Accessed 18 January 2021].

[6 Multiformats , «Mutlihash - Self-describing hashes,» Protocol Labs, [Ηλεκτρονικό].

Available: https://multiformats.io/multihash/. [Πρόσβαση 06 October 2021].

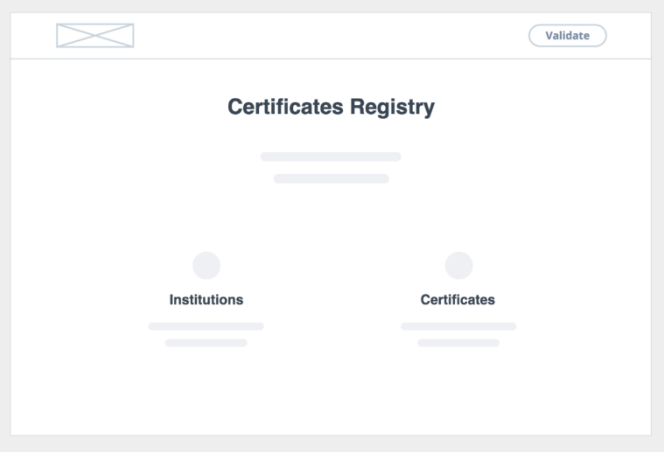
[6 International Telecommunication Union (ITU), "Statistics," [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx. [Accessed 30 December 2020].

[6 Web Design Museum, "Gallery of Web Design History," [Online]. Available: https://www.webdesignmuseum.org/exhibitions/web-design-in-the-90s/yahoo-1994. [Accessed 30 December 2020].

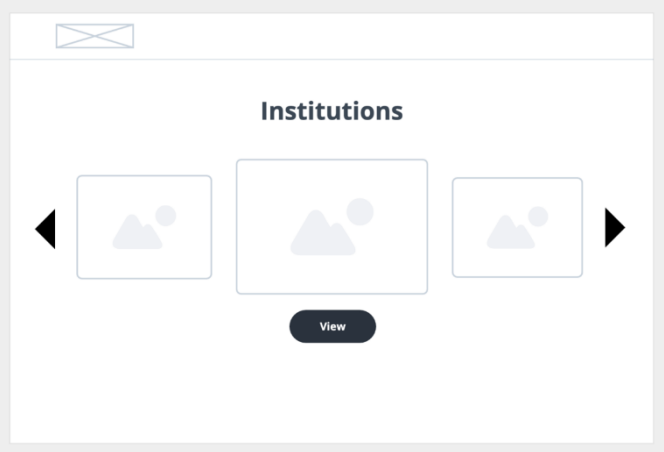
[7 A. Dhakal and X. Cui, "Blockchain and Smart Contracts for Internet of Things: A Systematic Literature Review," April 2018. [Online]. Available: https://www.researchgate.net/publication/332671231_Blockchain_and_Smart_Contracts_for_Internet_of_Things_A_Systematic_Literature_Review. [Accessed 02 January 2021].

[7 S. Sayeed, H. Marco-Gisbert and T. Caira, "Smart Contract: Attacks and Protections," January 2020. [Online]. Available: https://www.researchgate.net/publication/338926064_Smart_Contract_Attacks_and_Protections. [Accessed 06 October 2021].

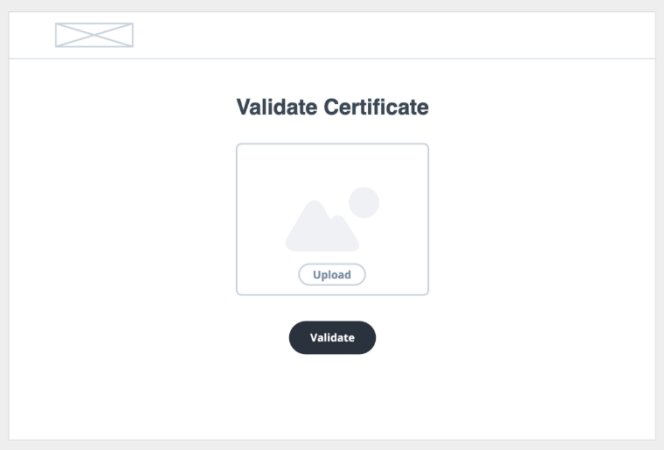
Appendix A CryptoCerts wireframes



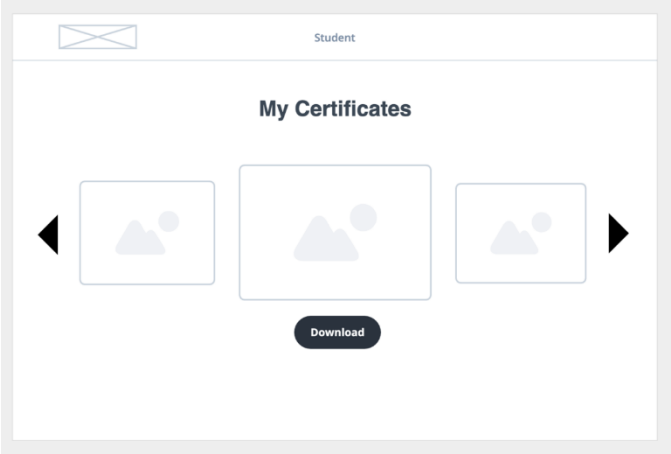
Wireframe 1 Home screen (Guest)



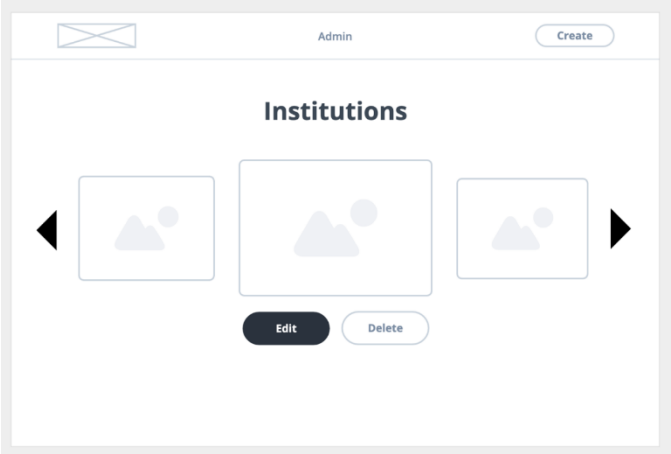
Wireframe 2 Institutions screen (Guest)



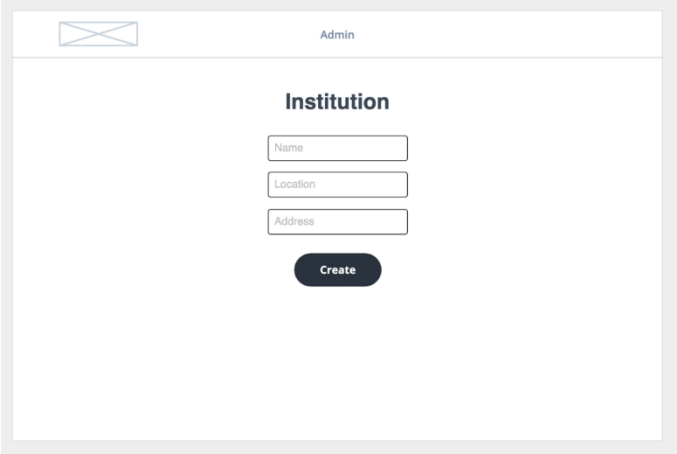
Wireframe 3 Certificate validation form (Guest)



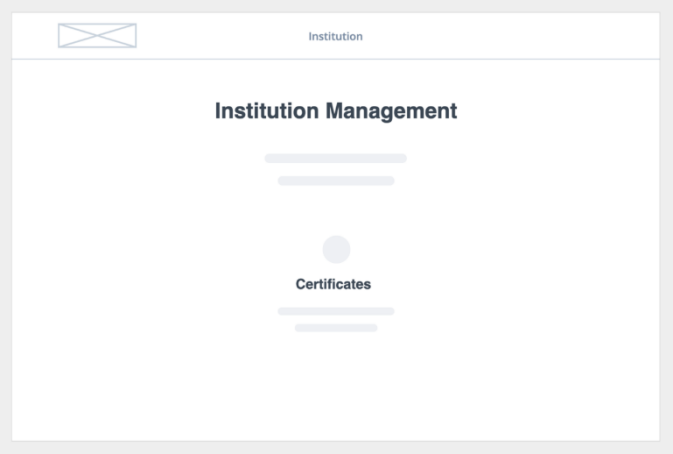
Wireframe 4 Certificates list (Student)



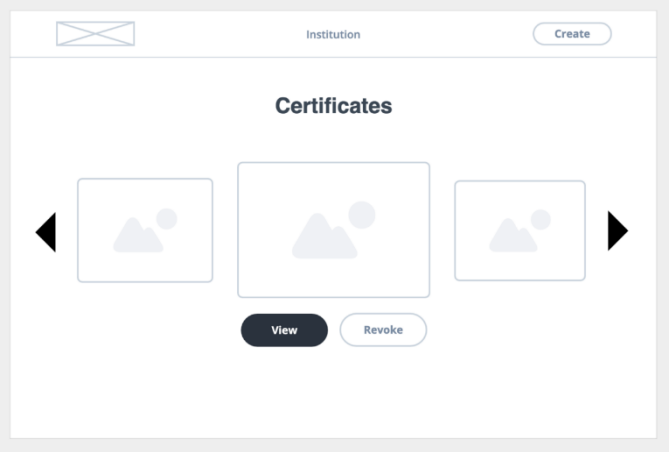
Wireframe 5 Institutions list (Administrator)



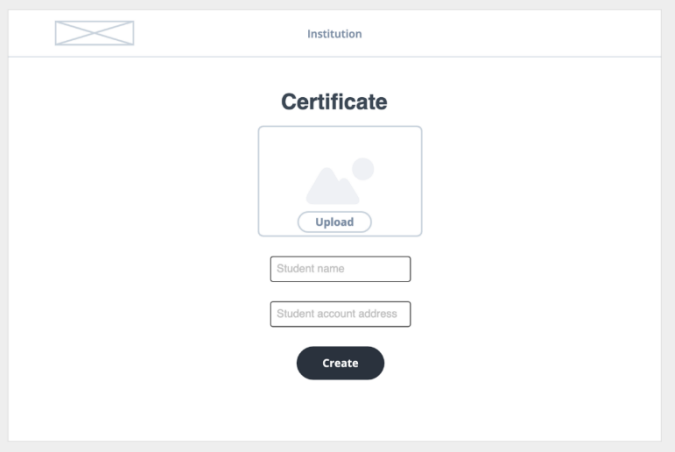
Wireframe 6 Institution form (Administrator)



Wireframe 7 Home screen (Institution)



Wireframe 8 Certificates list (Institution)



Wireframe 9 Certificate form (Institution)

Appendix B CryptoCerts source code

The code of the CryptoCerts decentralized application implemented as part of this dissertation is hosted in the following two repositories hosted on GitHub.com, along with detailed setup instructions:

- **cryptocerts-contracts** – https://github.com/lephleg/cryptocerts-contracts
- **cryptocerts-client** – https://github.com/lephleg/cryptocerts-client