Security and Privacy Evaluation of Blockchain Applications

a Thesis presented
by
Leopoldo Duenas Castro
to
The Department of Informatik und Elektrotechnik

in partial fulfillment of the requirements
for the degree of
Master of Science
in the subject of
Information Technology

Thesis advisor: Prof. Dr. rer. nat. Nils Gruschka
Second advisor: Dr.-Ing. Meiko Jensen

Fachhochschule Kiel: University of Applied Sciences
Kiel, November 2017

Statutory Declaration

I declare that I have developed and written the enclosed Master Thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master Thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Location, Date

Signature

Security and Privacy Evaluation of Blockchain Applications

Abstract

Most modern services and web applications are based on a centralized structure, requiring the intervention of a third party for their operation. Personal and sensitive information is shared with these services; therefore, users have to trust a proper management of their data. These centralized services often have a single point of failure which attackers can exploit to expose the application's data.

Blockchain technology incorporates a peer-to-peer architecture with no single point of failure. Furthermore, it provides certainty that an event or transaction took place without the need of an intermediary. The innovation behind the blockchain resides in the consensus mechanism it employs, along with cryptographic algorithms that secure operations within the network.

With the purpose of determining if a decentralized application can offer equivalent functionality as traditional web applications, a decentralized application (dApp) is built based on Ethereum's blockchain. Afterwards, it is evaluated in terms of functionality, security and privacy.

The results obtained show that a dApp can offer similar functionality as conventional web applications (with its performance limitations). Furthermore, this work describes the benefits regarding security (high Integrity, Non-repudiation and strong Authentication). Additionally, the authentication method offers a simplified mechanism from the end-user perspective. However, decentralized applications should follow security conventions and best practices to prevent attacks. Moreover, this thesis presents the privacy concerns derived from the blockchain's public nature.

Contents

List of Figures

List of Tables

Listings

Acknowledgments

First, I would like to thank my parents Leopoldo Dueñas Olmedo and Maria T. Castro Valenzuela for their support through this journey, also to my siblings Manuel and Estefania Dueñas.
Second, I would like to thank Prof. Dr. rer. nat. Nils Gruschka for his useful advice and support in the course of this work. Additionally, to Dr.-Ing. Meiko Jensen who provided helpful insight and input for the topic selection of this thesis.
Finally, to Satoshi Nakamoto, for introducing a new technological revolution.

1 Introduction

1.1 Motivation

The blockchain is a type of distributed network that serves as a database that contains historical records of all activities occurring within it. Despite being a relatively new technology, Blockchain has expanded itself from the original currency exchange domain to more sophisticated applications, such as smart contracts and decentralized applications in different fields (e.g., user Identification, Healthcare, E-voting).

Security and Privacy are two aspects often not a top priority in the development of an application. Popular services offered by companies like Google and Facebook are centralized, meaning users provide their personal information (passwords, names, birthdays, pictures) to these organizations. Users must trust a proper management of all their data to the third party.

There had been several known cases of user's data breaches in the last few years. Recently, Yahoo announced their security incident from 2013 was more prominent than initial claims on which all user email accounts were compromised (3 billion) [126]. Around 20,000 Tesco Bank customers were victims of an attack in 2016 on which money was stolen from accounts, in addition to banking details and personal customer information [90]. Deloitte's blue-chips branch was part of an attack were customer data was compromised, including usernames, passwords, health data and other personal information. The attack appeared to come from an administrator account with privileged access [100].

Future new developments may use the power of blockchain and its decentralized nature to solve problems like digital ownership of assets [83], ownership of personal data like healthcare records; a patient could use the blockchain to store personal health records and later share them with trusted doctors and institutions. Another similar scenario is digital identity, where users have control of their personal information.

1.2 Problem Statement

The vision is to create applications without a single point of failure where users don't have to give their data to an intermediary to access a service.

Nowadays, most web applications require users to share their credentials (usernames, passwords). Afterwards, companies store the information in their servers where data breaches can occur since they are often

targets of attackers. This issue directly affects users because their personal information might be published or sold.

A decentralized application based on Ethereum blockchain will be developed and evaluated in terms of security and privacy.

1.3 Goals

Blockchain technology will be reviewed in a detailed manner. This thesis aims to achieve the following goals:

- Review blockchain technology, current state, and applications based on it.
- Give a general analysis of blockchain technology regarding security and privacy.
- Build a decentralized application which uses the blockchain for user authentication. The application should provide a service: a rideshare platform. Users should be able to create, search and book a ride through the application.
- Evaluate the offered service in terms of security and privacy.

1.4 Thesis Structure

This thesis is composed of seven chapters, appendices, a glossary and references.

- Chapter 1 is dedicated to the introduction.
- Chapter 2 provides an introduction to blockchain technology giving detailed information of its functionality.
- Chapter 3 focus on current applications of the blockchain.
- Chapter 4 describes security and privacy aspects of the blockchain.
- Chapter 5 introduces the decentralized application and its implementation.
- Chapter 6 presents the results obtained and an evaluation of the application, including security and privacy aspects.
- Chapter 7 gives the conclusion of this thesis and possible future work and improvements.

It is true that contemporary technology permits decentralization, it also permits centralization. It depends on how you use the technology.

Noam Chomsky

2

Blockchain Foundations

This chapter focus on the theoretical foundations of blockchain technology. Section 2.1 defines a blockchain while section 2.2 explains the differences between centralized and decentralized systems (as well as distributed). Section 2.3 deals with agreement inside the blockchain, describing the most common mechanisms to reach consensus. Later, section 2.4 explains the cryptographic principles behind the blockchain. Section 2.5 details the transaction process and section 2.6 the block generation. Finally, the previous knowledge is arranged together in section 2.7 to build the blockchain.

2.1 Blockchain

The blockchain refers to a public record of all the transactions occurred within a network. The transactions are presented in chronological order in groups of *blocks*, each new block contains a reference to the previous block, thus, creating a chain of blocks. This structure of interrelated blocks is the reason behind the name of blockchain. A block to a blockchain would be the corresponding of a page to a book.

The blockchain is also referred as a *public ledger* since anyone can see all transactions in the underlying network. But this term is not only attributed to the public ledger but also to the technology behind the blockchain.

2.2 Centralization vs. Decentralization

A centralized system has a central node that communicates with all the other nodes. The central node is the most critical part of this system if it fails, the communication is no longer possible. On the other hand, a decentralized system does not possess a central node or authority. Several nodes handle the central node responsibilities (centralized system), thus, preventing a single point of failure.

In a distributed system, all nodes are equal and no hierarchy exists. A typical example of a distributed system is a peer to peer network. The decentralization concept is very often used instead of distributed, including blockchain literature. Figure 2.1 shows the standard representation of centralized, decentralized and distributed systems.

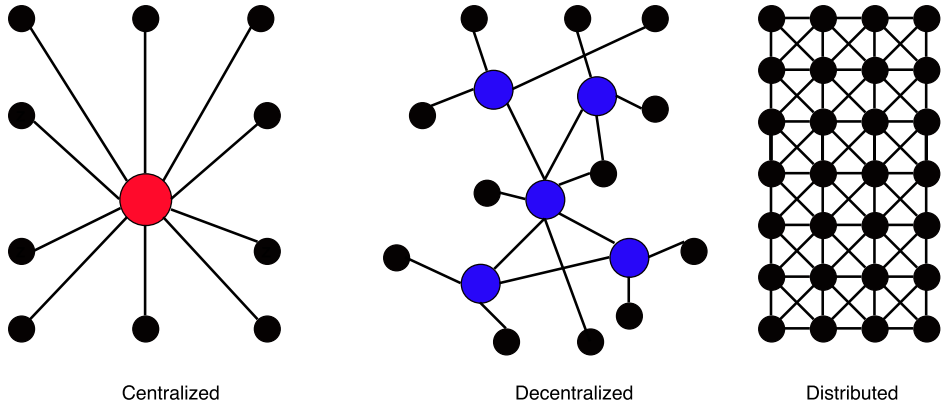


Figure 2.1: Centralized, decentralized and distributed representations.

The traditional economy functions in a centralized manner. If a person wants to send a transaction to another one, the first person usually requires the help of a central authority (e.g., a bank). The sender must provide to the third party the amount of the transaction plus any additional fees. The central authority handles the operation and sends the money to the second person. The sender and the receiver should trust the central authority to perform the transaction.

But not only belief during transactions is required in a traditional economy. Besides, citizens need to trust the central authority or bank the following aspects:

- The Issue of a valid currency.
- Effective management of all economic aspect of the currency.
- Keep an accurate balance of all customer's account.

As mentioned before, central authorities keep track of a client's account in the form of a private ledger. This account information should remain private between the central authority and the client. Since a bank has thousands or millions of clients, the number of accounts can be vast, and the maintenance of this kind of system can become complicated. Decentralized (distributed) systems follow a different approach: all the nodes on the network maintain one common ledger.

However, without a central authority in charge of handling the transactions backlog, this task becomes the responsibility of the nodes in a decentralized (distributed) system. Trust relies on reaching *consensus* between the nodes of the network in distributed systems.

2.3 Consensus inside the Blockchain

Consensus means reaching a collective agreement between the members of a network. Reaching consensus is an essential aspect in blockchain, since it enables the birth of new blocks and validation of transactions. The agreement allows the maintenance of a record that all the nodes support or at the merest they can recognize as legit [161].

Several different consensus mechanisms exist, including the following:

1. Proof-of-work.

2. Proof-of-stake.
3. Delegated proof-of-stake.
4. Other consensus mechanisms.

2.3.1 Proof-of-work

Proof-of-work was the first consensus mechanism incorporated in blockchain applications. Bitcoin, the first cryptocurrency uses proof-of-work. In proof-of-work, a certain amount of computer work must be performed to validate transactions. A requirement for networks based on proof-of-work is that most of the network's computing power should be concentrated in the hands of trusted nodes [132].

Bitcoin will be used as an example since it was the first implementation of blockchain technology. *Mining* is the process used to verify blocks and introduce the new currency to the Bitcoin network. Since no central authority has the responsibility of injecting cash into the system, this must be done in a decentralized manner.

A proof-of-work involves solving a mathematical problem that requires computing power; the total computing capacity of the *miners* defines the complexity. This difficulty is adjusted based on the average number of blocks solved per hour [132]. On average, a block is processed every 10 minutes in Bitcoin.

The mathematical problem involves finding a solution to an algorithm, in Bitcoin's case Hashcash [8]. This last one uses the *hash function* SHA-1, Bitcoin instead adapts SHA-256 [136] since it is stronger and much more *collision resistant* than SHA-1 [174]. SHA-256 always has the same length output, 256 bit (64 characters in hexadecimal), independent of the input size. SHA-256 is a one-way function, meaning it is easy to calculate but hard to reverse it [119].

The goal of proof-of-work is to find a hash value that begins with a certain amount of zero bits [132]. The computing power needed for its calculation increases with the number of zero bits specified. The difficulty is set by the Bitcoin protocol itself [9] and updated depending on the speed generation of the blocks.

The following is an example to illustrate the proof-of-work concept. The message "Decentralized applications" produces the following SHA-256 hash: 05c52912c16c6f0549ba3b323286f9c41379fd250b99c7bb1b1ed643dc65639c, the goal is to iterate and find a hash value starting with two zero digits by increasing a numerical value at the end of the message, starting with "Decentralized applications0". Table 2.1 shows the first iterations and the corresponding hash values. The first zero character is found in iteration #20 ("Decentralized applications19"), while the first hash value starting with two zero digits is iteration #194 ("Decentralized applications193").

This process is similar to the way blockchains adapt proof-of-work. However, the complexity in Bitcoin is many times higher since the computing power available is larger.

To obtain the desired hash value, miners vary a *nonce*. When a miner finally solves the mathematical problem, it receives a reward in the form of new currency.

Sometimes two miners will generate a block nearly at the same time; this is called a fork. In Bitcoin, forks are resolved by the rule of the longest chain [132], this means that the block referenced as the parent of the next block gets included in the principal blockchain. Figure 2.2 gives an example of a blockchain fork. Two

Message	*SHA-256 Hash*
Decentralized applications0	9971353d0c6b511ab706266fecadc84fd7b8d8711658d8baea827884cf2fcb9b
Decentralized applications1	9c186e3b9ccca3606d36808d26ede3a8d30554792f3a010f2c016872bcd9e69a
Decentralized applications2	ef70b3ccd596b51c0bf02db7be459f3f78330cb0223388eb1baff0149621ea9a
Decentralized applications3	4baf7b063547b64eb2f4034b5ae81d2f2e2e016dd611bd113ba97b797d3a4a87
Decentralized applications4	d9a4e7fa49e8e2a81df76ecb3976cac811b1d2a57916bdbad97e54421e1b8dba
...	...
Decentralized applications19	042a2265b99205aa28de10934521d7c9699e0deedfd118105dcb78c4665f25ea
...	...
Decentralized applications193	00fd17f5f06e6dabd1c3823d8a5a48c20e99dcb03840a4e14543bab15742af01

Table 2.1: Proof-of-work example using SHA-256

blocks #101 exist, but the first #101 block is referenced by the next block (#102), the second #101 block is discarded and the transactions included are returned to the queue of pending transactions.

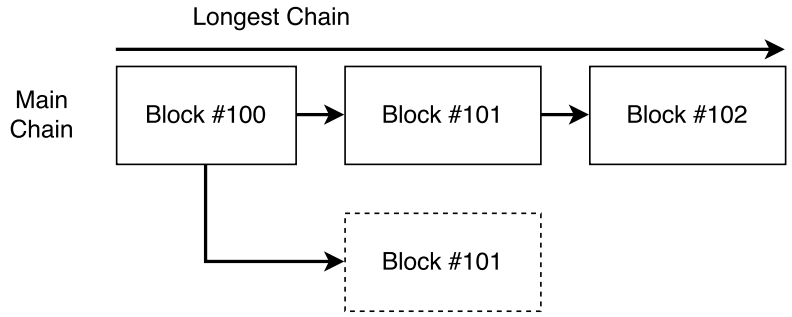


Figure 2.2: Blockchain Fork.

2.3.2 Proof-of-stake

A stake is an interest [39] or some property owned by a stakeholder. In blockchain technology, a stake can be the possession of *bitcoins* in Bitcoin or *ether* in Ethereum [75].

A proof-of-stake is a consensus mechanism on which the validator (miner in proof-of-work) is selected in a pseudo-random manner based on its stake ownership in the underlying network. In a different type of proof-of-stake mechanism, validators are chosen randomly and blocks are added to the blockchain based on votes. [77].

Whoever wants to participate is required to provide a certain amount of currency upfront or stake. Hence, the name of proof-of-stake. This deposit is retained for some time and penalized to malicious participants.

Some fundamental differences between proof-of-work and proof-of-stake include:

- As mentioned before, proof-of-work requires computation power to solve mathematical problems with the purpose of validating blocks. For this reason, proof-of-work uses large quantities of electricity and thus, it is expensive to maintain [77]. In contrast, proof-of-stake does not require the use of all this computation power.

- In proof-of-work, it is not needed the possession of currency to become a miner. Proof-of-stake miners need to make a deposit in the blockchain's currency unit.
- New coins are awarded to the winner nodes in proof-of-work. Transaction fees are granted in proof-of-stake.
- In proof-of-stake, the rewards received are proportional to the stake [18]. In proof-of-work, this depends on the computation power instead.

Some examples of cryptocurrencies implementing proof-of-stake are NxT [138] and Blackcoin [185]. Peercoin uses a variant of proof-of-work and proof-of stake [111] while Reddcoin focuses on the availability of user's currency in the network denominated Proof-of-stake Velocity [153].

Ethereum, currently the second most important cryptocurrency regarding market capitalization [48], has plans to move in the future to a hybrid proof-of-work and proof-of-stake mechanism. Casper is the name of this protocol [76].

2.3.3 Delegated proof-of-stake

Delegated Proof-of-Stake follows a more representative democratic approach. Blocks are conceived after a voting process takes place where designated witnesses produce blocks. Stakeholders agree in a similar voting procedure the parameters such as fees, block size, number of witnesses and other aspects [160]. An example of Delegated proof-of-stake is Bitshares [24].

A couple of differences between proof-of-stake and delegated proof-of-stake include:

- Delegated proof-of-stake claims to be more efficient because only a handful of trust witnesses oversee the block generation. This efficiency translates into lower fees and faster block generation [25].
- Since rewards are granted according to the stake in proof-of-stake, participants with a low stake receive small compensation. The efficiency in delegated proof-of-stake allows greater rewards even for small participants (shareholders).
- Delegated Proof-of-stake is less decentralized than proof-of-work since the block generation concentrates in few nodes.
- Delegated Proof-of-stake suffers from the same problem as a democratic system: lack of interest or participation. This indifference can cause the control to fall into a small group of stakeholders [34].

2.3.4 Other consensus mechanisms

In addition to the mechanisms explained before, several others exist, and many others are likely to appear in the coming years. A couple of other examples of consensus mechanisms are:

1. Practical Byzantine Fault Tolerance, e.g., Hyperledger [101].

2. Leased Proof-of-stake, e.g., WAVES [190].

3. Proof-of-Importance, e.g., NEM [134].

2.4 Blockchain and Cryptography

The last section explained how consensus works into the blockchain and therefore transactions are validated. It also introduced hash functions and their role in proof-of-work. This section deals with more cryptographic concepts which enable blockchain functionality.

2.4.1 Asymmetric cryptography

Asymmetric cryptography allows blockchain's users send and receive transactions through the network. Asymmetric cryptography introduces a key pair system: public and private key. A private key should be only known by the owner while the public key can be widely distributed. In asymmetric cryptography, also known as public key cryptography, any user can send encrypted messages, but only the proprietary of the private key can decipher them. Asymmetric key cryptography was introduced by Whitfield Diffie and Martin Hellman in 1976 [57].

The next scenario provides an example of how asymmetric cryptographic works. Scenario: User A and User B want to exchange a message using asymmetric cryptography, and they follow the next steps:

1. User B owns a key-pair generator which produces a public key and private key.

2. User B distributes the public key to User A.

3. User A generates a message and encrypts it using the provided public key.

4. User A sends the encrypted message to User B.

5. User B decrypts the message with the private key.

Figure 2.3 provides a visual representation of asymmetric cryptography.

2.4.2 Digital signatures

A very known implementation of asymmetrical cryptography is digital signatures. A digital signature seeks to prove the *authenticity* of the sender along with *non-repudiation* and *integrity* of the message. Section 4.1 presents these concepts in more detail. Using the same scenario in 2.4.1,the digital signature process commonly involves the following:

1. User A owns a key-pair generator which produces a public key and private key.

2. User A creates a message and generates a hash of it.

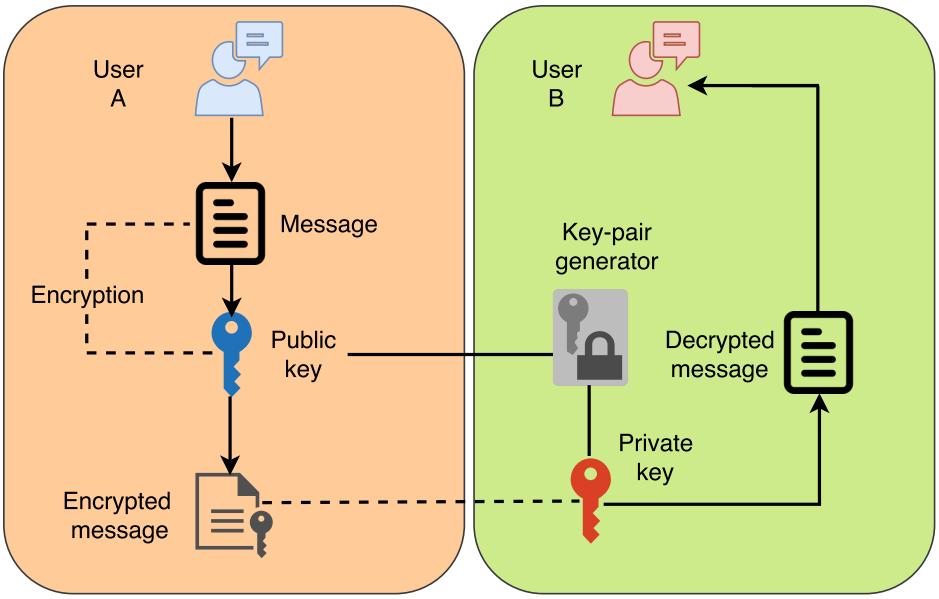


Figure 2.3: Asymmetric cryptography representation.

3. User A signs the resulting hash using the private key.
4. User A sends the message along with the signature (signed hash) and the public key.
5. User B receives the public key and the signed message. User B calculates a hash using the public key.
6. User B compares the calculated hash with the one provided in the message. If they match, authenticity of the sender and integrity of the message are validated.

Figure 2.4 presents a digital signature process.

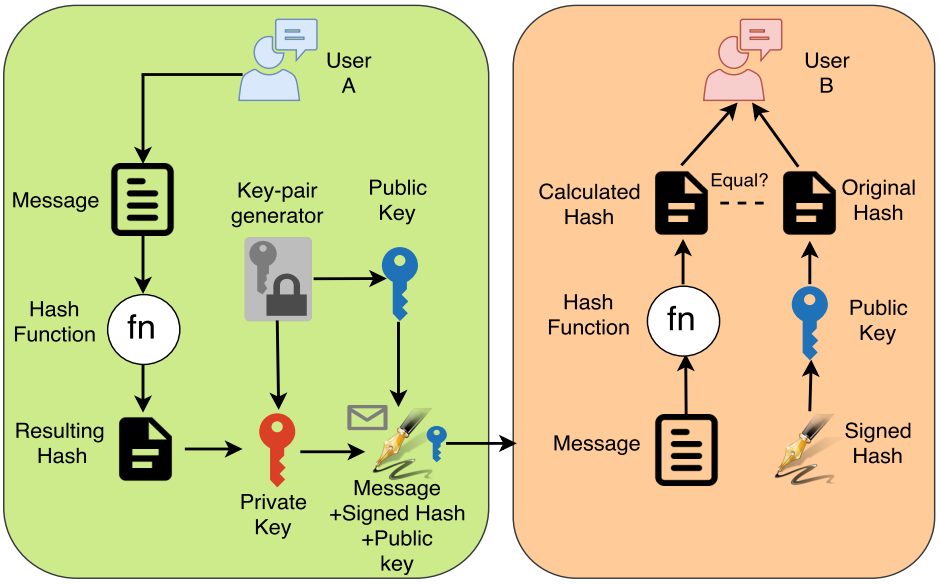


Figure 2.4: Digital signature process.

RSA, a public key cryptography algorithm is often used in digital signatures. RSA was first proposed in 1978 by Rivest, Shamir, and Adleman [155]. Bitcoin uses the Elliptic Curve Digital Signature Algorithm [105], which is also based on public key cryptography. An advantage of the Elliptic Curve Digital Signature Algorithm (ECDSA) is the length of the keys is smaller compared to RSA, achieving the same level of security [109].

2.5 Blockchain Transactions

Like a traditional wallet, Bitcoin wallet holds a user's assets. A wallet is composed of addresses and one or more private keys. An address serves as an attribute to identify transactions; users can deposit bitcoins or receive bitcoins through addresses. A different address should be used for each new operation due to the public nature of the blockchain; a user transaction record can be built from a given set of addresses since all transactions are public.

The private and public key generation involves the usage of ECDSA: a random secret key is selected and then a corresponding public key derives from the private key. Later, an address is generated from the public key using a double hashing of SHA-256 and RIPEMD-160 [146] algorithms. Finally, Base58Check encoding converts the address to a human readable format. Figure 2.5 shows the address generation process.

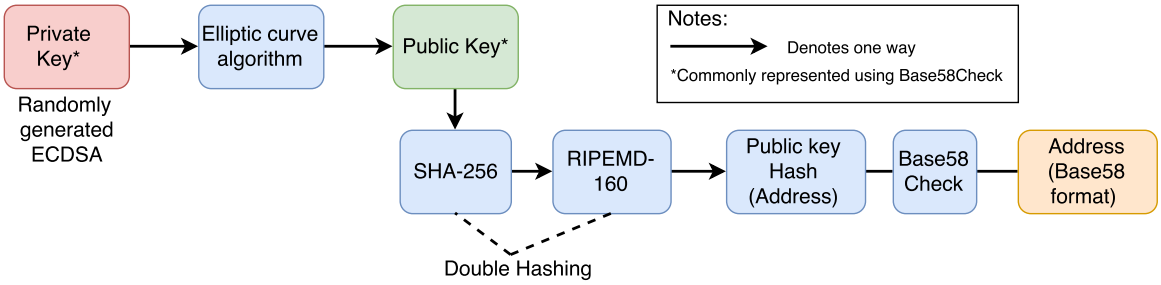


Figure 2.5: Bitcoin's public key and address generation from a private key.

There exist different types of transactions in Bitcoin, the most common transaction type is when a user transfer bitcoins to a second user. A transaction is composed of one or more inputs and outputs. A transaction input always comes from an output of a previous transaction; thus, the currency is always spent. An exception is "coinbase" transactions because they introduce new coins to the blockchain. If a user possesses two bitcoins and wishes to send one bitcoin, two outputs will be generated: the first, a one bitcoin spent and the second, containing the remaining unspent bitcoin minus fees.

Transactions fees are entirely optional in Bitcoin. However, they are encouraged since miners will put a low priority on transactions without any compensation. Figure 2.6 provides an example of the interaction of inputs and outputs in Bitcoin transactions. An input can generate one or more outputs, but an output can only be referenced by one input.

An example of two Bitcoin transactions will be presented in a synthesized form with the following scenario:

- User A sends "3" bitcoins to User B.
- Later, User B forwards the same "3" (minus fees) amount of currency to User C.

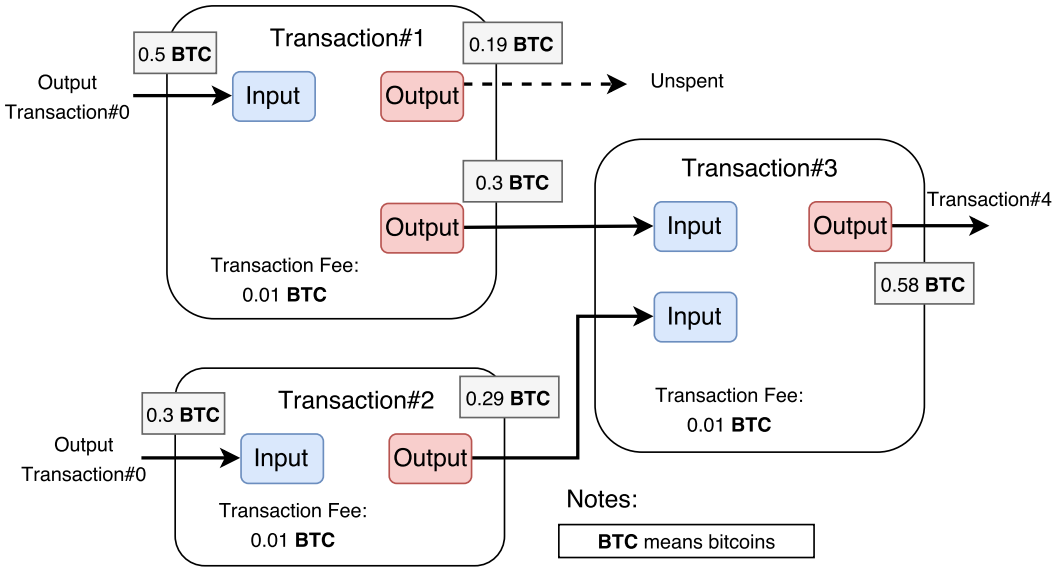


Figure 2.6: Inputs and Outputs in Bitcoin Transactions.

First Transaction: User A to User B.

1. The three users should have Bitcoin wallets. User A requires the destination address, a bitcoin address is currently between 26-35 characters long and begins with a digit "1" or "3" [17].
   For example:
   User's A address: 1B3VyXpCcM2V3EbWFoDfgHtnp2g9ZpcxW9,
   User's B: 1DByeF6CVpDaW2Q3zH54CXsQwzHK2wNTpV and
   User's C: 1Lg9pTkFvrpUVrL6HgFtn41m6vQ3Zomu2i.

2. User B gives User A a destination address which is the result of User's B public key hash (encoded in Base58 format), at the same time, this hash is derived from User's B private key, as presented in figure 2.5.

3. User A retrieves User's B public key hash from the provided address.

4. User A generates a new transaction with a value of "3" bitcoins (minus fees) to User's B public key hash. The transaction output can be spent only by the owner of the private key which is referenced by User's B public key.[13]. The parameter 'scriptPubKey" specifies the instructions to spend the output bitcoins.

5. User A creates a digital signature using his private key on his wallet [4]. Steps 3, 4 and 5 are handled by a Bitcoin client.

6. User's A transaction is released to the peer to peer network. After some time, User's A (minus "3" bitcoins) and User's B (plus "2.99" bitcoins) wallets will be updated reflecting the new balances.

Second Transaction: User B to User C.

1. User C gives User B a destination address.

2. User B retrieves User's C public key hash from the provided address.

3. User B generates a new transaction using as input the operation User A created previously. Additionally, User B creates a signature script which contains User's B private key signature and the respective public key "unhashed" [13]. When this public key is hashed, the resulting hash can be compared to the public key hash provided by User A (reminder: User's A public key hash was given by User B).

4. User's B transaction is released to the peer to peer network and miners can add it to their blocks. Any of the nodes on the network can verify User's B transaction by employing the corresponding public key. Miners form a block with the recent transactions.

5. Mining process takes place where miners try to obtain the proof-of-work to submit the next block. User's B transaction gets included in blocks.

6. Once a miner successfully solves the proof of work problem, the resulting block is added to the blockchain, and the miner receives the reward and transaction fees.

7. Before the block is incorporated to the blockchain, the other nodes on the network verify the block. Section 2.6 reviews this process.

8. User's B transaction is completed. Subsequently, User's C can spend the transferred bitcoins.

Figure 2.7 shows the workflow of the two transactions just described.

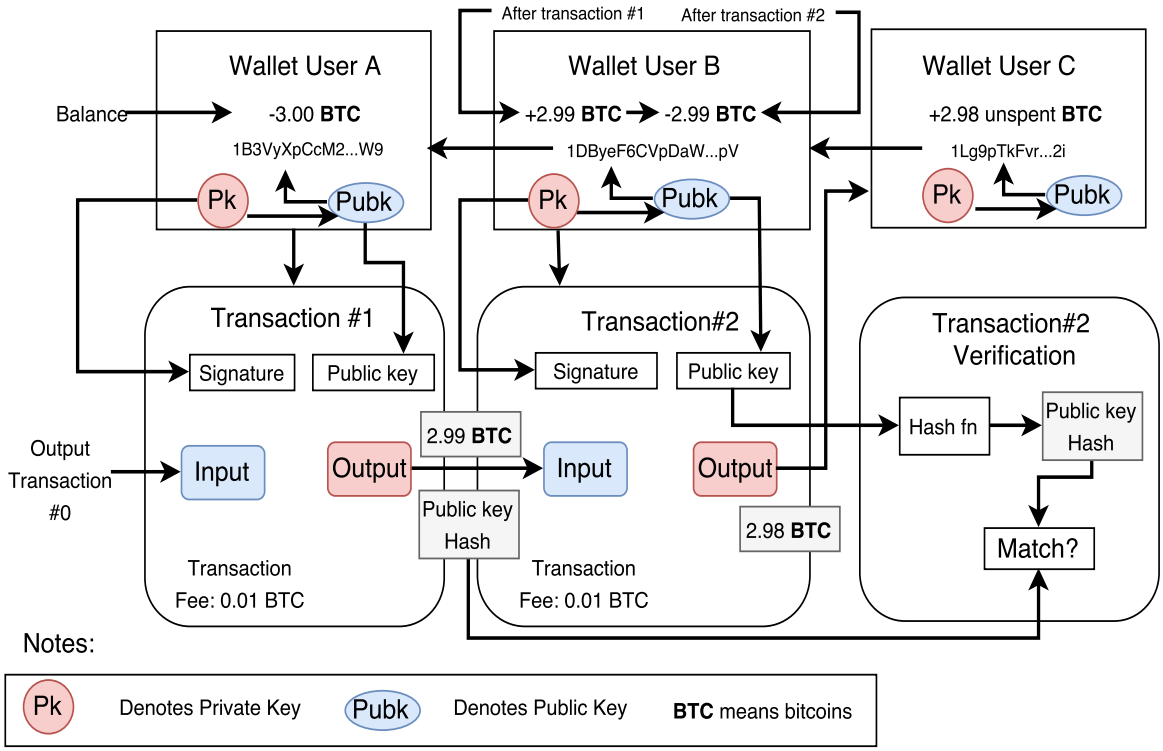


Figure 2.7: Example of Bitcoin Transactions.

2.6 Block generation

Several important aspects of the blockchain have been covered so far, including consensus mechanisms, transactions, decentralization and cryptography principles. As mentioned before, a block is the basic structure of the blockchain. Blocks sequence one after another to build the public ledger. This section deals with the construction of these blocks.

A group of transactions built by a miner form a block, the limit size of Bitcoin's blocks is 1 megabyte. However, miners decide the number of transactions included on each block, only limited by the 1-megabyte size.

A block is always linked to a parent block and composed of the following fields:

- *Block size*. The size of the block.
- *Block header*. Detailed information about the block.
- *Transaction counter*. The number of transactions in the block.
- *Transactions*. All the transactions compromised by the block.

Since the block header (80 bytes) gives specific information about the block, it is formed by the fields described in Table 2.2.

Size	*Name*	*Description*
4 bytes	Version	Describe the software version.
32 bytes	Previous block hash	The hash value of the previous block header.
32 bytes	Merkle Root	The hash value of the block's Merkle tree, see section 2.6.1
4 bytes	Timestamp	Approximate time when the miner created the block.
4 bytes	nbits	Refers to the difficulty goal of the block (target proof of work hash value)
4 bytes	Nonce	The nonce generated by the miner of the block.

Table 2.2: Block Header

There are two forms to identify a block: the block height and the block hash. The block height refers to the block's serial number, the equivalent of the page number in a book. The first block created or genesis block has a block height of "0". On the other hand, the block hash is the result of double hashing the block header using SHA-256 [4].

Table 2.3 is an example of a block header from Bitcoin's genesis block, block height *0* and block hash *000000-000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f* [29].

The nbits parameter is a compressed representation of the target proof of work value of the block. The nbits field provides the target's bytes length on the first byte and the most significant bytes of the target on the last three bytes, figure 2.8. The target is a 256-bit number [13].

The nBits value of Bitcoin's genesis block is 1d00ffff in big-endian notation. Form this value, the target proof of work can be deducted as follows:

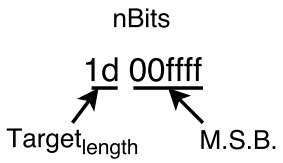


Figure 2.8: nBits parameter to target of proof of work.

1. Target Length is given by:

$$T_{length} = 0x1d \tag{2.1}$$

("1d" equals 29 in Decimal)

2. Target Most Significant bytes:

$$T_{MSB} = 0x00ffff \tag{2.2}$$

3. Multiply T_{MSB} by the base 256 squared by exponent 0x1d minus the bytes in T_{MSB} [107]:

$$T = 0x00ffff * 256^{(0x1d-3)} \tag{2.3}$$

4. The result is the target value:

$$T = 00000000ffff0000000000000000000000000000000000000000000000000000 \tag{2.4}$$

Figure 2.9 gives a graphical representation of this conversion between nbits compressed target value to the equivalent 256 bits target proof of work of genesis block.

	Target = 256 bits = 32 bytes (hexadecimal)																														
	*	*	*	Target_length = 0x1d (29 bytes)																												
Bytes				**1**	**2**	**3**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Hex	00	00	00	**00**	**ff**	**ff**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
				M.	**S.**	**B.**																									
Note:	* Additional zeroes to complete the 256 bits target value.																														

Figure 2.9: Deduction of Bitcoin's genesis block target proof of work value from nBits parameter.

Field	*Value Hexadecimal big-endian*	Description
Version	01000000	Version 1
Previous block hash	0000000000000000000000000000000000000000000000000000000000000000	First block, thus no parent.
Merkle Root	4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b	Merkle root hash
Timestamp	495fab29	Unix: 1231006505
nbits	1d00ffff	Target proof of work
Nonce	7c2bac1d	2083236893 (Decimal)

Table 2.3: Bitcoin's Genesis Block Header

2.6.1 Merkle Trees

The Merkle root of the block is the product of the hashes of a block's transactions. Bitcoin uses Merkles trees[128] for two purposes: first, verify that a particular transaction is included in the block and second, ensure the integrity of the transactions contained in the block. If the order of transactions change or a transaction information is altered, the Merkle root value changes as well.

Nodes on the network first verify transactions, including inputs, outputs, signatures and their correctness, as mentioned in section 2.5. This process prevents the double spending of currency. It is possible for a miner to avoid this verification process; however, this block would later be rejected by the other nodes because it doesn't fulfill the requirements.

The process to create a Merkle tree is the following:

1. Transactions are hashed using SHA-256.
2. Transactions hashes are paired in groups of two. If an odd number, the remaining transaction hash is duplicated and matched to itself.
3. The groups of two transactions hashes are then hashed together.
4. Step 2 and 3 are repeated until one hash is obtained: the Merkle root. This parameter is assigned as the "Merkle root" in the block's header.

Figure 2.10 is a Merkle tree representation of four transactions.

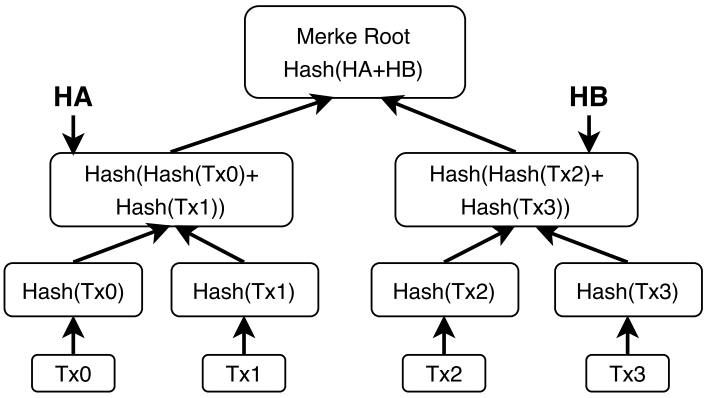


Figure 2.10: Merkle tree of four transactions.

Simple Payment Verification, often denominated SPV, are nodes that don't collect the entire blockchain, but instead, they use Merkle trees to validate if a transaction exists in a block. SVP, store the block headers only. A transaction can be validated by using its hash and then asking for other necessary hashes of the block to a node. In the example of figure 2.10, the hashes of Tx2, HA and the Merkle root (which the node SVP already knows) are necessary to validate transaction Tx3. Table 2.4 gives an example of the number hashes (excluding the Merkle root) required to verify a transaction belongs to a block [4]. The number of hashes needed to prove a transaction membership in a block increases in a much lower rate compared to the number of transactions.

Number of Transactions	*Block size approximate*	*# Hashes*
4 transactions	1 kilobyte	2 hashes
16 transactions	4 kilobytes	4 hashes
512 transactions	128 kilobytes	9 hashes
2048 transactions	512 kilobytes	11 hashes
65,535 transactions	16 megabytes	16 hashes

Table 2.4: Hashes required to validate a transaction using Merkle Trees [4].

2.7 Building the blockchain

So far, different aspects of how blockchains work have been reviewed. In summary, the process of a transaction submission and verification to the blockchain includes:

1. *Transaction creation by some user*. See section 2.5.

   (a) Users should possess an amount of currency, for example, a Bitcoin address inside a wallet containing bitcoins.

   (b) Users exchange addresses. The first user generates a new transaction using a client. The public key of the receiver is extracted from the provided address and the first user signs the operation with its private key. The sender's unhashed public key is likewise attached to the transaction.

   (c) The transaction is submitted to the network.

2. *Block generation and Consensus*. See section 2.6.

   (a) Miners (Bitcoin/Proof of Work) take recent transactions from a pool and start to build a block. Transactions are verified and hashed together forming a Merkle tree.

   (b) Miners compete against each other to solve the proof-of-work and obtain a block header hash lower than the target, see section 2.3.1. In this step, the nodes on the network achieve consensus. Alternative consensus mechanisms to proof-of-work exist, see section 2.3.

   (c) When a miner solves the proof-of-work, the block is transmitted to the other nodes in the network who verify it.

   (d) If the block is valid, a reward plus transactions fees are granted to the creator of the block. The block is finally added to the blockchain.

3. *Transaction completion*

   (a) After the generation of the block containing the transaction, the currency can be spent by the recipient.

(b) The transaction information is publicly available.

Figure 2.11 gives a visual representation of this process.

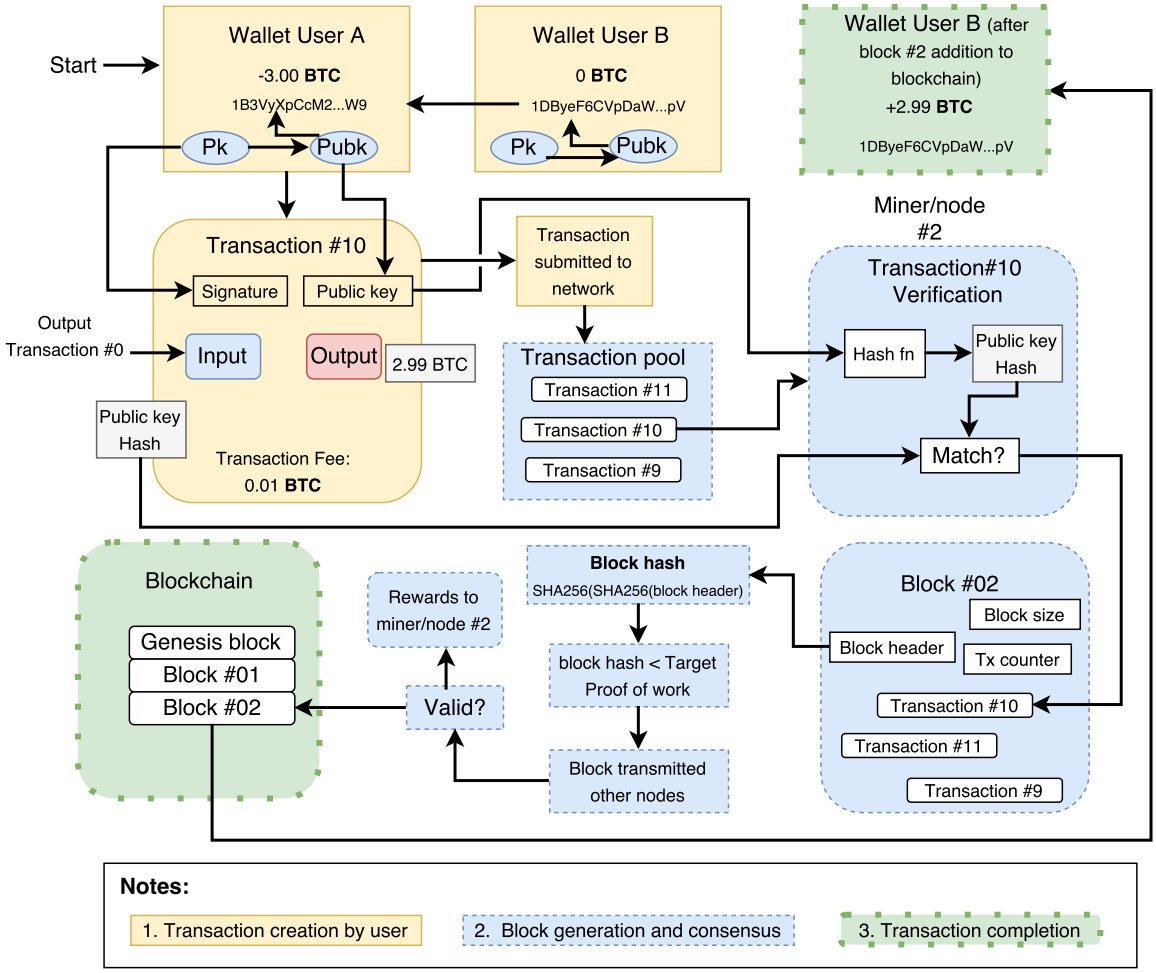


Figure 2.11: Blockchain complete transaction process.

While technology is important, it's what we do with it that truly matters.

Muhammad Yunus

3 Blockchain Applications

This chapter reviews different blockchain applications. The purpose is to establish a background for this thesis by discussing the importance of cryptocurrencies, smart contracts, decentralized applications and other technologies developed using the blockchain. Three sections are presented as follows: Section 3.1 reviews cryptocurrencies. Section 3.2 looks into smart contracts, Section 3.3 deals with decentralized applications and further blockchain applications.

3.1 Cryptocurrencies

The term cryptocurrency alludes to a digital currency which uses cryptography to control the origination of units of currency and ensure transactions are valid. Cryptocurrencies are the first and currently most known application of blockchain technology. Cryptocurrencies involve different advancements to function, including: Cryptography, peer to peer networks and decentralized systems.

In 1981, David Chaum published a paper titled "Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms" [40]. This document was one of the earliest to present a methodology to achieve some degree of pseudonymity in financial transactions.

Dwork and Naor first introduced the idea of proof-of-work by establishing the requirement of a computation work to make use of a resource, with the objective of combating junk mail [61]. In 1997, Adam Back introduced Hashcash [8] a proof of work algorithm. Later Jakobsson and Jules formalized the concept of Proof of Work [104].

One of the firsts attempts to create a digital currency was E-gold, it allowed users to trade gold and other precious metals as currency [205]. Napster[133], a known pioneer of peer-to-peer technology appeared in 1999. All these technologies were predecessors of Bitcoin, the first Cryptocurrency.

In 2008, Bitcoin was introduced in the paper "Bitcoin: a Peer to Peer Electronic Cash System" [132]. The following year, Bitcoin officially launched. The innovation behind Bitcoin lays in its architecture where users confirm transactions without the need of a third party.

Currently, Bitcoin is the most popular cryptocurrency in terms of market capitalization [48]. However, other cryptocurrencies have appeared following the steps of Bitcoin. As of 2017, there are over 800 cryptocurrencies [48]. Table 3.1 contains the top 10 most popular cryptocurrencies based on market capi-

talization (October 7, 2017).

Cryptocurrency	*Market Capitalization (USD)*	*Currency value (USD)*
Bitcoin	$72,066,919,849	$4,339.11
Ethereum	$29,101,601,611	$306.35
Ripple	$9,042,847,322	$0.23
Bitcoin Cash	$6,005,487,052	$360.14
Litecoin	$2,753,402,367	$51.69
Dash	$2,318,150,426	$304.74
NEM	$1,888,506,000	$0.21
NEO	$1,695,215,000	$33.90
IOTA	$1,379,544,809	$0.49
Monero	$1,364,944,994	$89.89

Table 3.1: Top 10 Cryptocurrencies by Market Capitalization [48]

3.1.1 Bitcoin

Bitcoin opened the door for the development of cryptocurrencies. Although the market capitalization of Bitcoin (around $72 billion dollars [48]) is hefty, traditional digital payments such as PayPal and credit cards trade even higher numbers.

In 2016 they were a total of over 82 million transactions with an average of 2.62 transactions per second, estimates based on [31]. In the same period, PayPal had around 193.4 transactions per second [143] and VisaNet around 1,736 transactions per second [186] (table 3.2).

Payment method	*Transactions per year*	*Transactions per second*
Bitcoin	82 million [31]	2.62
PayPal	6.1 billion [143]	193.4
VisaNet	54.75 billion [186]	1736

Table 3.2: Bitcoin, PayPal and VisaNet Transactions per year and second in 2016

Figure 3.1 compares the transaction volumes of Bitcoin, PayPal, and Visa in 2016. Although transaction volume of Bitcoin has increased extremely fast in the last few years, from 4.3 billion in 2015 to 58 billion in 2016 [32], still not possesses a trading volume as significant as PayPal.

Despite being not a preferred transaction method, Bitcoin and cryptocurrencies have become an alternative payment method. Also, their importance cannot be denied in today's economy.

Bitcoin currently faces several challenges, including the following:

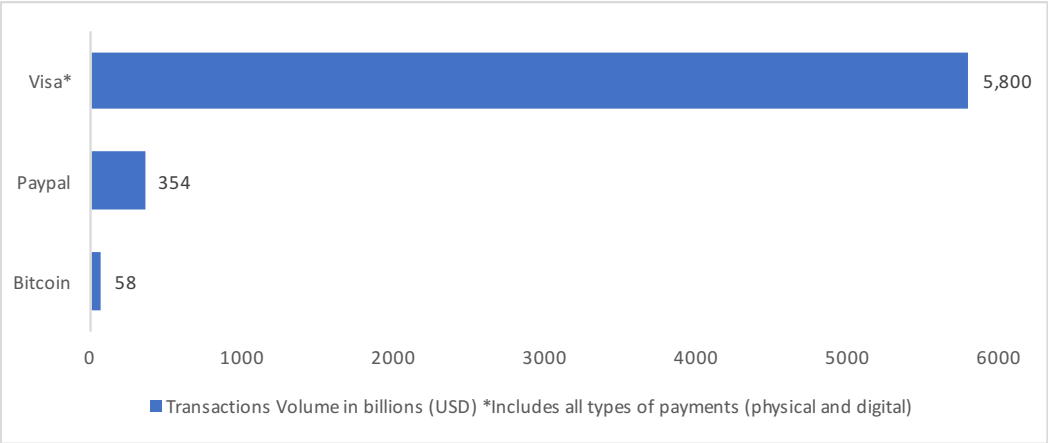


Figure 3.1: Bitcoin's, PayPal's and Visa's Transaction Volume in 2016.

Scalability

As Bitcoin increases its popularity, transaction numbers have raised at the same time. A difference between Bitcoin and other digital payment options such as PayPal is that the latter can handle thousands of transactions per second. However, Bitcoin is limited to blocks of 1-megabyte size. Therefore, the maximum number of transactions per second is approx. 7 [19]. Figure 3.2 shows bitcoin's block size as of 2017, the size has steadily increased and almost reaching the limit size. [28].

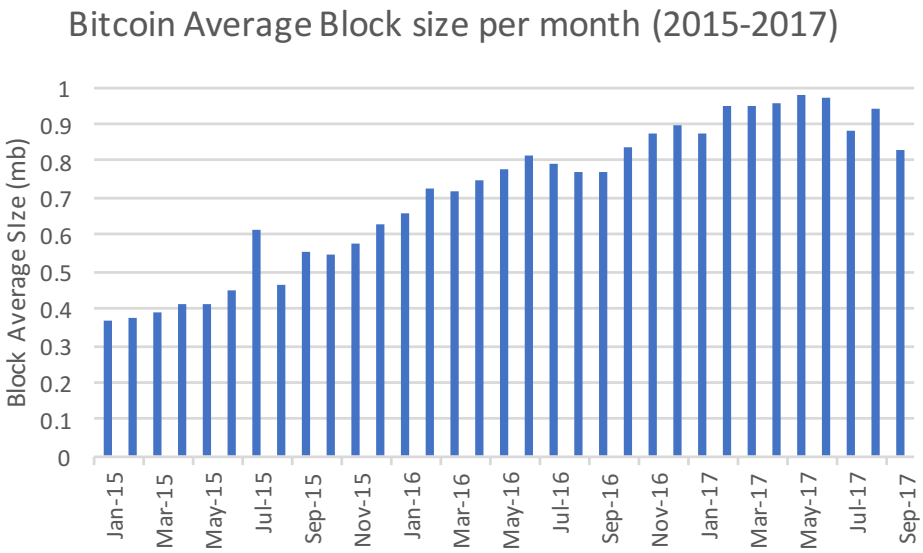


Figure 3.2: Average Bitcoin's Block size per month (Jan 2015-October 2017) [28].

Proof-of-work allows reaching consensus about the state of the blockchain. However, agreement about Bitcoin's network parameters, such as block size, mining difficulty, rewards and others, had been a continuous debate in Bitcoin's community.

Two possible solutions gained traction in 2017: *Segwit2x*, a two-step update which started in August of 2017. The first step launched Segregated Witness, this protocol in simple terms moves the signature script to an extended block, thus increasing the blocks size [169]. However, the block size remains at 1 megabyte

since the signature script does not count within this limit. Another benefit of this signature and transaction data separation is protection against transaction malleability. The second step, expected for November of 2017, is the increase of block size to 2 megabytes.

The second solution is *Bitcoin Cash*, a Bitcoin hard fork that was timed to match Segwit2x rollout in August of 2017. This Bitcoin fork created a new currency: Bitcoin Cash(BCH). Bitcoin Cash is a duplicate of Bitcoin blockchain (before the fork), however, after the fork, both blockchains became independent of each other. The supporters of Bitcoin Cash believe Segwit2x is not a sufficient solution to Bitcoin's scalability issues. For this reason, Bitcoin Cash implements a maximum size of 8 megabytes for each block [16].

Updates to Bitcoin's network

Reaching consensus about the changes to the Bitcoin's network parameters is complicated, as mentioned previously. Due to this reason, any update of Bitcoin's rules takes time to implement. Other blockchains have proposed using the same principles of blockchain technology to manage these changes. Delegated Proof-of-Stake, for example, seeks to follow a voting process within the blockchain to agree about the network parameters [160].

Proof of Work is expensive

Proof-of-work requires a lot of computation power to process transactions in Bitcoin. As of October of 2017, the current hash rate is around 8,600,000 TH/s (trillions of hashes per second) [30].

Claimed as one of the most efficient mining equipment, Antminer S9 has a power efficiency of 0.098 W/GH/s [22]. By multiplying these two numbers, the energy consumption is given. The overall consumption of Bitcoin's blockchain is currently about 843 megawatts. Since a household in the US spends in average 29.62 kW/h per day [182], Bitcoin energy consumption will be the equivalent to power around 680,000 American households (considering every miner is using one of the most efficient mining equipment), following the calculation method from [124].

One of the possible solutions to reduce the electricity used is the increment of the block size. This action would allow to include more transactions per block and thus reducing the energy spent per transaction. As mentioned before, there still debate of the best direction to follow.

In summary, Bitcoin faces several challenges and currently one of the most important is scalability. This issue should be solved for further adoption of this blockchain application.

3.1.2 Ethereum

Although Ethereum is not a big as Bitcoin in terms of market capitalization, currently Ethereum market is around $29 billion US dollars [48], becoming the second largest cryptocurrency. Ethereum is relatively new compared to Bitcoin; it was first proposed by Vitalik Buterin in 2013 and later released in 2015 [65].

Ethereum seeks to expand blockchain applications beyond currency, thus introduces smart contracts to blockchain technology. Section 3.2 provides more specific details about smart contracts. Ethereum intro-

duces the Ethereum Virtual Machine (EVM) which allows the execution of code [64], so users of Ethereum blockchain can order the execution of their defined algorithms [200].

Several differences exist between Bitcoin and Ethereum blockchains. Table 3.3 compares the two of them in a summarized manner.

Characteristic	*Bitcoin*	*Ethereum*
Currency	bitcoins (BTC)	ether (ETH)
Application	Digital Transactions	Digital Transactions, Smart Contracts, Decentralized Applications
Currency Supply	21 million BTC	No limit (steadily created)
Consensus Mechanism	Proof-of-Work	Proof-of-Work. Plans to move to an hybrid Proof-of-work/Proof-of-Stake mechanism [76]
Hashing Algorithm	SHA-256	EtHash
Exchange method	Addresses	Accounts
Transaction Fee currency	BTC	Gas (ETH)
Mining reward	12.5 BTC (Halves every 210k blocks)	5 ETH (may be reduced to 2 ETH in upcoming update)
Block maximum size	1 megabyte	Variable (defined by the block's gas limit, variable value)
Block time	10 minutes	15-17 seconds [89]

Table 3.3: Comparison between Bitcoin and Ethereum blockchain

While Bitcoin employs addresses to transfer funds from one user to another, Ethereum introduces accounts for this activity. There are two kinds of accounts: *Externally Owned Accounts* and *Contracts Accounts*. Externally Owned Accounts can transfer ether and call the execution of contracts. Contracts Accounts possess executable code that runs when other contracts or transactions call it [62].

Bitcoin uses SHA-256 for Proof-of-Work and Ethereum Ethash instead, SHA-256 requires computing power (CPU) and Ethash RAM. Therefore, Ethereum encourages mining through GPU's rather than CPU's [159].

The block size in Ethereum is based on the block's gas limit. Currently, this value is approx. 6,693,522 gas [79]. Another difference between Bitcoin and Ethereum are how transaction fees are handled, in Bitcoin transaction cost directly BTC but in Ethereum they cost gas. Gas is required for any transaction that interacts with the blockchain. There are several concepts associated with gas in Ethereum [63], including:

- *Gas Price*: The equivalent cost of gas in another currency, e.g., bitcoins, ether.
- *Gas Limit*: A variable value that specifies the top quantity of gas used in a block. It serves as the block size limit.
- *Gas Cost*: The price to execute an action in the blockchain in units of gas. This value does not change over time.

- *Gas Fee*: The transaction fees of either a currency exchange or contract execution.

A substantial difference between Bitcoin and Ethereum is the transaction time. A Block creation in Bitcoin takes around 10 minutes, in comparison, it takes approximately 15 seconds in Ethereum [89]. In Bitcoin, mined blocks created slightly after a new block is added to the blockchain are discarded. Therefore, the computation power used to calculate the proof-of-work is wasted.

Ethereum denominates the discarded blocks "uncle blocks" and they have a value. Miners are encouraged to include a mention of uncle blocks in their blocks; every included uncle block makes their chain gain weight. While in Bitcoin, the longest chain is included to the blockchain, in Ethereum is the "heaviest" [197].

The reason to include the uncle blocks is that they have a certain amount of work performed by miners. This amount of proof-of-work aims to increase the security of the blockchain, more proof-of-work in a block increases the computation and therefore the cost to reverse it. It also reduces the waste produced by proof-of-work mechanism. This feature allows Ethereum to issue blocks in a shorter time without making it too easy for attackers to double spend or reverse transactions.

Two future updates are scheduled for Ethereum:

1. *Metropolis*: Aims to introduce substantial changes to Ethereum blockchain in two phases:

   - *Byzantium*. Expected at block 437,000 (October 2017) [80], this update seeks to improve smart contracts and introduce Zk-Snark or Non-interactive zero-knowledge proof [33]. The aim of the adoption of Non-interactive zero-knowledge proof in Ethereum is to allow private transactions between users.

     Recently, block generation has slowed down going from an average of 15 seconds at the beginning of 2017 to almost 30 seconds in October [82]. Figure 3.3 graph shows the average block generation time from January to October of 2017. Byzantium will postpone the difficult bomb, which goal is to increase the mining difficult and thus ending Proof-of-work by making it too expensive. Another important update is the reduction of block reward, from 5 to 3 ETH [180]. This reduction could translate to faster and cheaper transactions according to O'Leary [139]. Additionally, contracts will have the option to pay the transaction fees.
   - *Constantinople*. There is not yet a release date, but expected at some time of 2018 [80]. This phase should prepare further for the migration to Proof-of-Work/Proof-of-Stake mechanism (Casper), as well as improve functionality after the upgrade to Byzantium.

2. *Serenity*: This is a relevant update which is the last of four significant upgrades planned for Ethereum. Currently, two have been already released, with the third one being Metropolis and the last one Serenity. This update aims to introduce Casper and thus migrate from Proof-of-Work to a hybrid Proof-of-Work and Proof-of-Stake consensus mechanism. Serenity will provide developers more options and advanced application of smart contracts. Blockchain "sharding" is also included on this update, sharding term is referred to the partition of the blockchain in smaller pieces and distributing each piece to a node, such feature should help Ethereum network become faster and reduce block times [165].

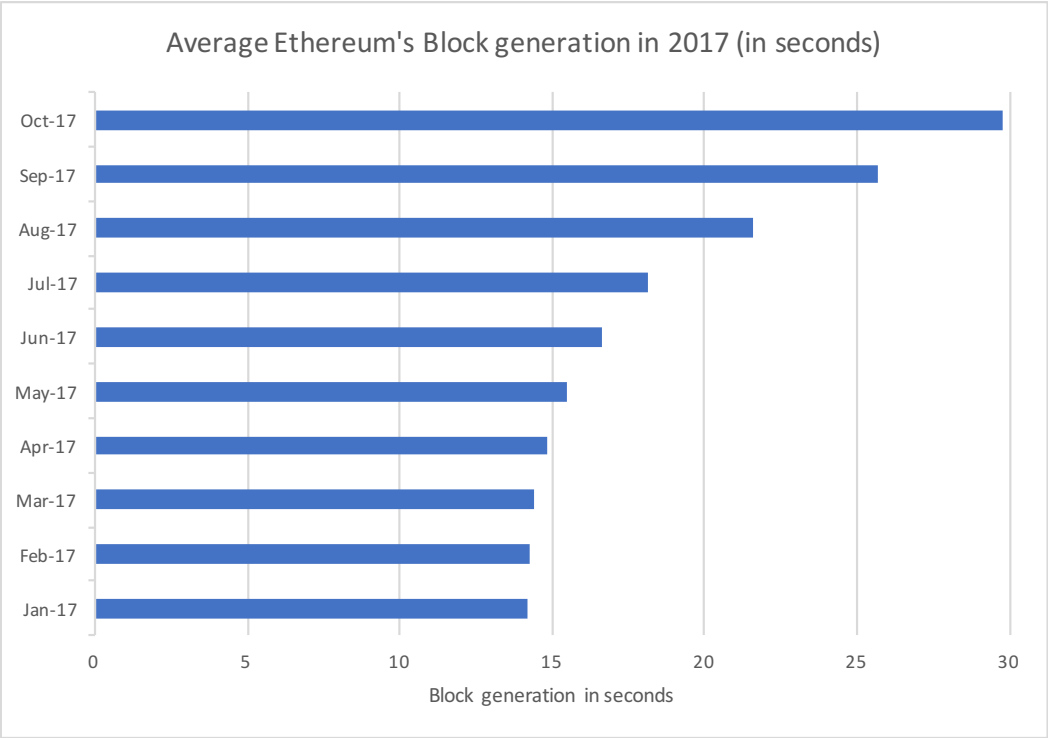


Figure 3.3: Average Ethereum's block time in seconds (2017) [82].

3.2 Blockchain 2.0 Smart Contracts

Smart contracts are often denominated as Blockchain 2.0, as they extend blockchain functionality beyond currency exchange applications. A contract is an agreement between two interested parties or more. A contract can be legally binding, meaning it can be enforced and all the parties should comply with all the agreements specified in the contract. Usually, a third party with authority execute a legally binding contract.

Some examples of contracts include one person buying a land, asking for a loan to a bank or getting married. When an individual (borrower) signs a loan with a bank (lender), both parties acquire responsibilities. The lender should provide the loaned money to the borrower, while the latter should reimburse the agreed reimbursement, loan amount plus interests, to the lender within a predefined period. With these types of contracts, there is usually a guarantee like a mortgage or other possession. If the loaner does not comply with his part of the agreement, the lender can take legal actions and force the loaner to provide the guarantee. In this case, a third party is required to enforce the contract.

The term smart contract was proposed by Nick Szabo [173]. The idea behind smart contracts is that a middleman or third party is not required to execute or enforce a contract. As of today, smart contracts can perform the next activities [98]:

- A smart contract functions as an intermediary between two or more users. The contract enforces the agreements/actions stated in it.
- Save application data, e.g., for a decentralized application or *dApp* .
- Serve as a service for other contracts.

- A type of contract called "wallet contract" can be executed when the specified number of signatures is reached [81].

Figure 3.4 provides a representation of a contract and a smart contract.

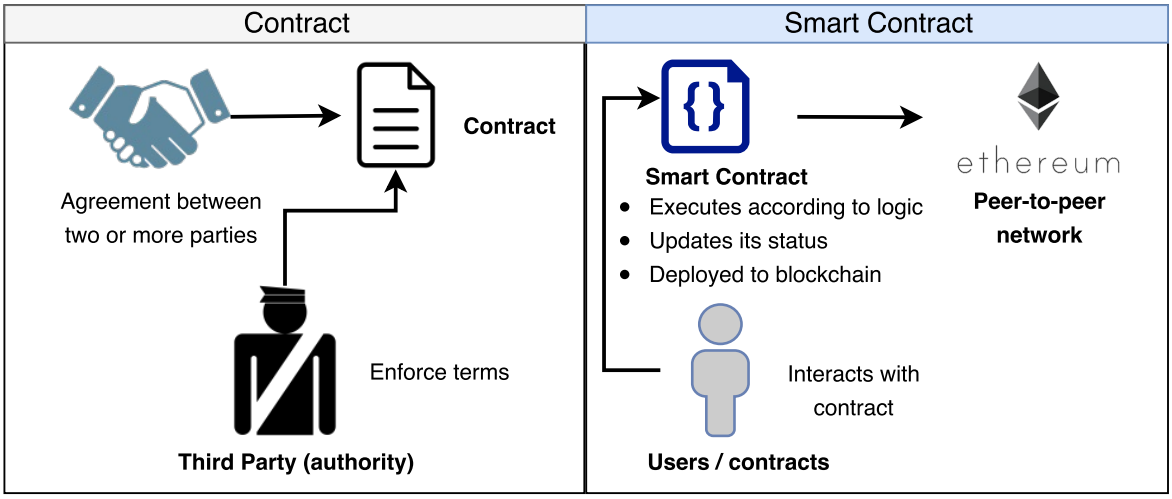


Figure 3.4: Representation of a contract and a smart contract.

3.3 Blockchain 3.0 and Decentralized Applications

Blockchain 3.0 refers to applications that extend the scope of blockchain technology even further. A decentralized application (dApp) doesn't have a central server handling application logic, opposed to centralized applications. The dApp concept has been popularized thanks to Ethereum platform and its support for this type of applications. However, decentralized applications existed before, e.g., BitTorrent, a peer-to-peer system to share different kinds of files [42], Tor, a network composed by virtual tunnels which goal is to provide more privacy for internet activity [59]. As just reviewed, decentralized applications are not limited to blockchain technology. Nevertheless, for the scope of this work, a decentralized application or dApp concept will be used specifically for blockchain applications.

In a simplified manner, a typical web application is composed of the following elements:

- *GUI (Graphical User Interface).* Users directly interact with the GUI.
- *Server.* The server handles user request. The server and GUI communicate together based on user's input.
- *Database.* Contains the application's data. The server requests information from the database and or writes new data.

On the other hand, a simple dApp structure is given by:

- *GUI (Graphical User Interface).* Same as a typical web application, handles user interaction and displays the requested content.

- *Blockchain node.* A node to connect and interact with the blockchain.
- *Smart Contracts.* Handle application's logic.
- *Blockchain/Public ledger.* Smart Contracts are deployed to an underlying blockchain, where they are publicly available.

As can be seen, the main difference between a smart contract and a decentralized application is the user interface presented in dApps. Figure 3.5 illustrates a simple web application and one decentralized application. As shown, the decentralized application has no central server.

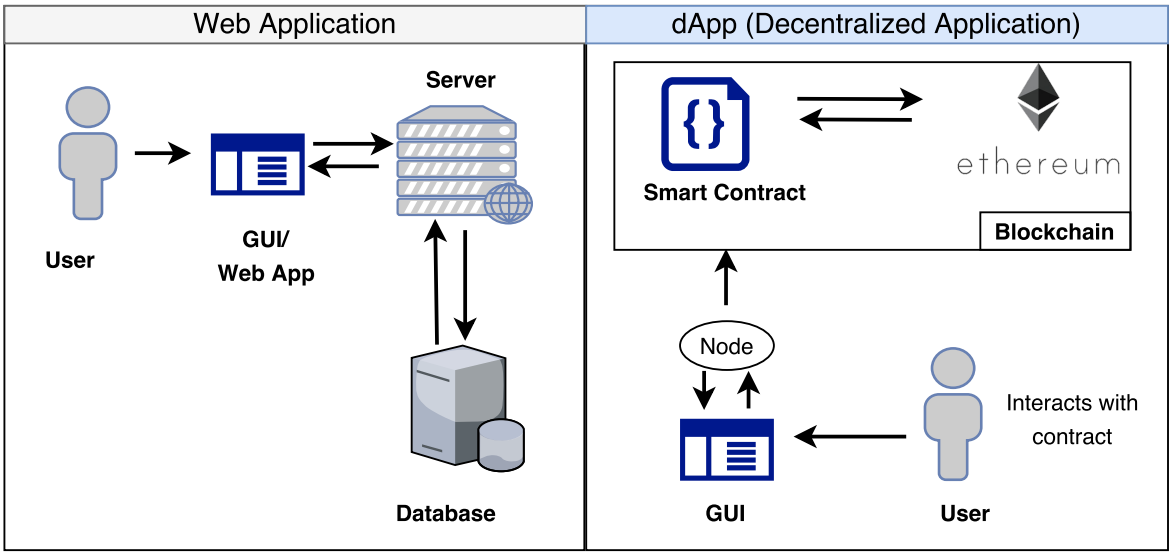


Figure 3.5: Comparison of a simple web application and decentralized application (dApp).

3.3.1 Decentralized Applications

Ethereum is a vital source of decentralized applications. Currently, there are over 700 applications listed in Ethereum [171]. Some of the applications are under development, others are prototypes, demos, concepts or finalized versions. Table 3.4 provides a list of relevant dApps.

3.3.2 Further Blockchain Applications

As reviewed in the last section, blockchain is being used in many new fields. The following list presents some interesting blockchain applications currently in development.

- Distributed cloud Storage
  - *Storj.* An open source distributed cloud storage, it provides end to end encryption, allowing users to pay for the required storage they need. On the other hand, users get paid for renting their own hard drive space. The documents uploaded to Storj are divided into pieces, then encrypted and finally dispersed through the network. Storj uses the blockchain to store file integrity and location data [195].

dApp	*Type*	*Status*	*Description*
Block Action [27]	Financial	Live	Digital wallet for transactions with Bitcoin and Ethereum without third parties.
ICO Wizard [198]	Business	Live	Creation of crowd-sales campaigns.
Trippki [176]	Business	Work in progress	Hotel booking service with rewards system.
Numerai [52]	Business	Live	A hedge fund that seeks to create a bridge between machine learning and the stock market by overcoming overfitting.
Kript [114]	Business	Work in progress	A mobile application for trading and investing in cryptocurrencies.
HelloSugoi [102]	Business	Live	A platform for buying and selling tickets for different types of events.
Stockblock [172]	Business/Legal	Prototype	Marketplace and copyright protection for artwork.
Legatum [118]	Legal	Work in progress	Store documents such as legal papers or of intellectual property to the blockchain. File stamping proves ownership of property.
Network Democracy DAO [150]	Governance	Prototype	A governance/democratic system for voting proposals
Storj [195]	Technology	Live	A distributed (peer-to-peer) cloud storage platform. Allows to rent free hard drive space.
Dentacoin [55]	Healthcare	Live	Review platform for dental treatments with the goal of achieving better quality and service to dental patients.

Table 3.4: List of relevant Ethereum Decentralized Applications (dApps)

- *Sia*. Sia claims to be around ten times cheaper than other centralized solutions, such as Amazon S3 [163]. Although following the same principles as Storj, Sia does not run in an existing blockchain. Sia has its own peer-to-peer network, which is used to distribute the uploaded files [187].
- *SAFE Network* SAFE (Secure Access For Everyone) network [123] uses the same functionality principles as Storj and Sia, calling the process of storage and data distribution "self encryption" [122]. A notable difference between SAFE and the prior systems is the scope, SAFE plans to focus not only on storage but also offer applications, messaging, email, social networking and other services [123].

• Identity Management

- *ShoCard*. This company aims for people to have control of their own identity. The process consists of uploading/scanning an identity document, extraction of personal information, data encryption, followed by data hash/signature creation and later storage in the blockchain. [162].
- *Uport*. An open source project that using Ethereum blockchain for the creation of a "self-sovereign identity" system. A mobile app, currently Alpha version allows users to create identities, manage credentials and sign transactions within Ethereum blockchain [121].
- *Blockstack*. Blockstack is an open source project which allows the creation of decentralized applications. It offers storage connection to cloud services (e.g., Amazon S3, Dropbox, Microsoft Azure, and others) and authentication solutions. Blockstack runs on top of a blockchain, but without depending on a specific blockchain [2]. For example, Blockstack could move from Ethereum to Bitcoin or the other way around.
- *Bitnation*. Bitnation Public Notary (BPN) intent is to provide the same services as a traditional government, such as ID issuance and notary services like marriage contracts, birth certificates, business contracts, land titles and many others. The blockchain stores the records of these services [23].
- *E-residency*. E-residency is a program from Estonia government which uses smart cards for signing documents, non-residents from Estonia can apply. The aim is to facilitate entrepreneurs to open a business in Estonia without the need to be physically located in the country [154].

• E-voting

- *Follow my vote*. Follow my vote uses the blockchain technology for online voting. Since all transactions are contained in the public ledger, the verification of all the casted votes is transparent [88].
- *BitCongress*. A legislation and voting system based on blockchain technnology[157].

These are only a couple examples showing some of the potential applications of blockchain technology. Additional applications could further expand the potential of the blockchain, such as Decentralized Organizations (entire companies running in a decentralized form).

The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts.

Gene Spafford

4

Security and Privacy in the Blockchain

The goal of this chapter is to introduce the security and privacy aspects of the blockchain. Section 4.1 of this chapter presents security and privacy basic concepts. Section 4.2 covers security of the blockchain and section 4.3 deals with privacy.

4.1 Security and privacy concepts

4.1.1 Security

Security can be defined as *"protection against threats"* [36]. Two parts of this concept can be further explored: the object to protect and the threat. The object to safeguard can be anything that has a value: a person, possession (a car, a house), a country, information, etc. The field of security depends on the type of object being protected, for example:

- A person: Human/Personal security.
- A possession: Security of goods.
- A country: National Security.
- Information: Information Security.

An asset is often referred as the object to protect *asset*. For information security, an asset can be emails, company and customer data, sensitive information and other types of digital data.

A threat is anything that *"presents a danger to an asset"* [194]. Following the previous example of fields of security, table 4.1 presents some threats.

Information Security is defined by the National Institute of Standards and Technology (NIST) as the perseverance of *"information and systems from unauthorized access, use, disclosure, disruption, modification, or destruction to provide confidentiality, integrity, and availability"* [135].

The last three mentioned concepts within this definition form the CIA triangle (figure 4.1), often referred as the goals or objectives of computer security.

Asset	*Security Domain*	*Threats*
Person	Human/Personal Security	Diseases, accidents.
Possession	Security of goods	Thieves, accidents.
Country	National security	Terrorism, espionage, natural disasters.
Information	Information security	Hackers, viruses, malware.

Table 4.1: List of assets and corresponding threats.

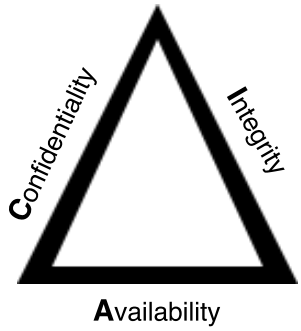


Figure 4.1: CIA triangle: Confidentiality, Integrity and Availability.

The following elements composed the CIA triangle:

1. *Confidentiality*: The goal is to provide access to information only to the individuals who have the appropriate rights [194]. Information has different levels of confidentiality, depending on the type of information. For example, a purchase receipt at a grocery store has not the same level of confidentiality as a social security number, an address or credentials to access a web application.

2. *Integrity*: Ensure information is preserved in its original form without changes by unauthorized sources. There are two aspects to consider about integrity; the first one is the preservation of unaltered information, often denominated data integrity. The second refers to the source of information [12], whoever or whatever is the author of the data. An example can be a letter sent to a company where the signature is modified. In this case, the data integrity remains since the message didn't change. However, the source of the data is altered, and therefore the integrity of the letter is compromised.

3. *Availability*: Reliable access to information whenever needed, promptly and in the required presentation [135]. If for example, a student wants to borrow a book from the school's library, but some other student already took the book. Then, the information is not available since he or she can not access the resource.

Table 4.2 presents several scenarios where information has been compromised. The respective goal of security is mentioned in every case.

Non-repudiation is another essential security concept. Non-repudiation goal is to ensure that the sender of a message is the actual author of it. Moreover, the sender can confirm the receiver obtained it [170].

Access Control grants access to a system or resource to authorized users while at the same time preventing unauthorized access. Furthermore, access control restricts what users can and cannot do with the system or

Scenario	*Security goal*	*Explanation*
User A wants to send an email to User B. However, he forgot his password, and he is not able to restore it.	Availability	User A can not access a resource as needed.
Worker A has a job in a company and sends customer information to a third party. Worker's A company signs non-disclosure agreements with all customers.	Confidentiality	Worker A shared confidential information to unauthorized sources.
Worker B is a writer in a newspaper and he published a story, changing the name of his source.	Integrity	The source name changed.
A company's authentication system that handles and stores employees accounts has been hacked. The attacker removed access to the accounting database.	Confidentiality, Integrity and Availability	Confidentiality: Information should not be accessible to the hacker. Availability: attacker removed access to a database. Integrity: attacker modified the system by removing access to the database.

Table 4.2: Information security scenarios and their respective security goals

resource. For example, an employee in a company may be allowed to read a database while a manager may have additional writing rights for the same resource. Usually, access control has three parts:

- *Identification*: The process of establishing the identity of something or someone. For example, when a person meets someone and provides his name. In a computer, the equivalent of a username.
- *Authentication*: The process of demonstrating the claim of being something or someone [131]. A person can affirm being someone, but that does not mean the statement is true. An example of authentication is providing a driver's license or passport to prove identity. In digital terms, a corresponding password. Different authentication methods exist, usually classified into three categories:
  - Ownership (something the user possesses). Examples: Identity cards, passports, keys.
  - Knowledge (something the user knows). For example: passwords, personal identification numbers (PINs), security questions, passphrases [10].
  - Inherence (something the user is). Examples: Biometric (fingerprint, voice, retina, facial), signatures.
- *Authorization*: The process of allowing access to a resource after successful authentication [170]. After authentication, the user should have access according to his rights.

Large quantities of data are easily accessible online thanks to the increased usage and global adoption of the internet. According to the Cisco Visual Networking Index (VNI) the IP traffic, consumer plus business, is expected to almost triple from 2016 to 2021 [41]. Figure 4.2 presents the IP traffic projections from 2016 to 2021.

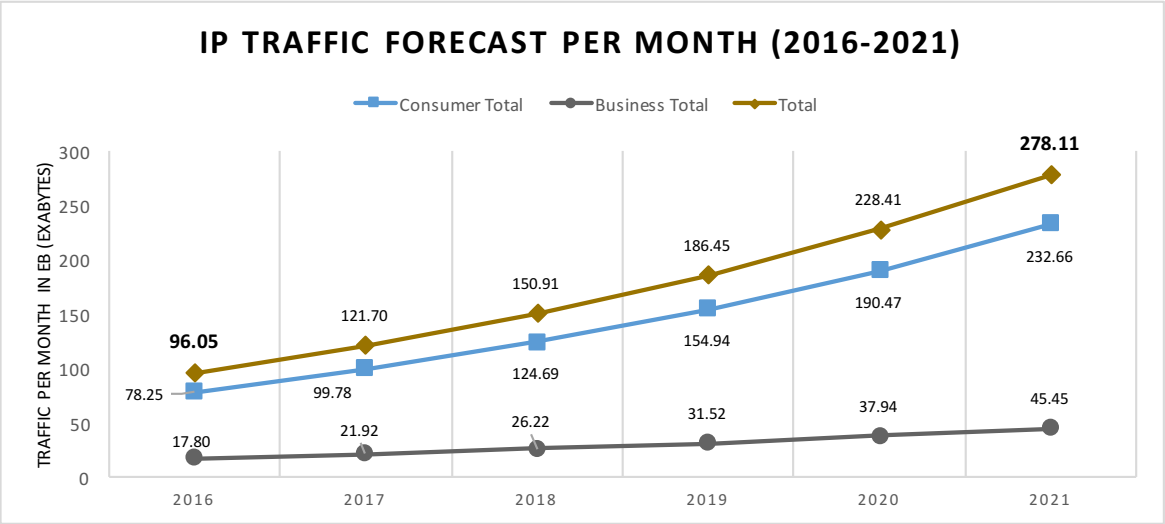


Figure 4.2: VNI's IP Traffic projections per month (2016-2021) [41].

A vulnerability is any flaw in a system that may open the door to attacks [194]. In the modern era, many cases of data breaches have occurred. In 2014, eBay was the victim of a cyber-attack which resulted in over 145 million user's accounts compromised. The attackers gained access to the user's database utilizing corporate employees accounts [86]. The FriendFinder network exposed over 412 million user accounts due to a security vulnerability. It was also informed passwords were either stored in plaintext or using SHA-1 algorithm [56]. The SHA-1 hash function is no longer considered secure and was deprecated by the National Institute of Standards and Technology (NIST) in 2011 [137].

One of the most popular cases is the Yahoo data breach on which all Yahoo accounts (including email addresses, passwords, names, dates of birth) were stolen in 2013, affecting over 3 billion users [126]. A previously known data breach had already taken place in 2014.

A vulnerability affecting the most of Wi-Fi enabled devices was published in 2017 [184]. The issue resides in the 4-way handshake (defined in 802.11i), which according to Vanhoef and Piessens is susceptible to a key reinstallation attack. Most Wi-Fi enables devices are affected by this issue, including Apple, Android, Windows, OpenBSD, MediaTek and Linux devices. Attackers may intercept a network's traffic and perform other malicious attacks by exploiting this vulnerability; this is a relevant discovery since Wi-Fi protocol has been considered secure for a long time.

Due to the importance of protecting information against attackers and vulnerabilities, information security plays an essential role in this digital era.

4.1.2 Privacy

The concept of privacy is a broad term, for example, Warren and Brandeis defined it just as *"the right to be let alone"* [58]. While Bustin described privacy as *"the claim of individuals, groups, or institutions to determine*

for themselves when, how and to what extent information about them is communicated to others" [193]. The concept from Bustin alludes to personal information which includes, for example, the following:

- Personal information: name, gender, marital status, birthday, age, social security number.
- Contact information: home address, work address, email addresses, phone numbers.
- Biometrics: fingerprints, retina, facial.
- Sexual orientation.
- Health condition: diseases, allergies.
- Political view: political affiliations.
- Personal records: criminal, driving incidents, infractions.
- Economical: monthly income, taxes.
- Digital information:
  - MAC addresses.
  - Emails sent/received.
  - Localization: smartphone sensors data, GPS localization.
  - Social media: pictures, messages.
  - Account usernames and passwords.
  - Other applications data: bank accounts, shopping records.

A good part of this information is sensitive, and for this reason, it should be protected. At first, it may seem simple to keep personal information safe by just not sharing this data with anyone. However, this is impractical since sharing information is needed for everyday life. For example, when a new employee begins working in a company, providing personal and contact information is required. For the registry of most services, this data is necessary, e.g., contracting utility services.

Although security and privacy are closely related, they are not the same. Security allows users and systems to access data if they have the proper rights, privacy goes beyond this by allowing the access of "what" and "when" needed [54].

For instance, medical employees should have access to health records of a patient, security deals with who should be able to read the patient's records, privacy deals with what information of the patient should be revealed and under which conditions. An employee should not be able to review records if, for example, the patient is not currently treated.

Security is required to preserve data, e.g., medical employees in a hospital have access to a patient's records. However, it is not enough to deal only with security [103]. Medical employees should not access the patient's data whenever they want or for personal purposes. Therefore, privacy is desired.

4.2 Blockchain security

Since the initial application of blockchain technology is the digital exchange of currency, security is an important aspect to consider. As reviewed in Chapter 2, the combination of the consensus mechanism (e.g., proof-of-work) along with cryptography principles (hash functions, asymmetric cryptography, digital signatures) is the base of blockchain security.

Diversity is good for the blockchain; as the number of honest nodes participating in the mining process increases, the chance of a successful attack reduces or becomes more difficult [110]. In proof-of-work, computation power is required to double spend currency, therefore, when the sum of nodes increases, an attack becomes more expensive. However, this does not mean an attack is impossible.

Although volatility on the exchange cost of cryptocurrencies is high, and therefore a risk of cryptocurrencies, in 2017 the overall price of popular cryptocurrencies has increased. Figure 4.3 shows the average price per month of Bitcoin and Ethereum from January to October of 2017 [45, 46]. One bitcoin at the beginning of 2017 was worth around $900 US dollars, increasing to almost $5,000 US dollars on average in October. In the same lapse, one ether went from around $10 US dollars to nearly $300 US dollars. Blockchain presents an undeniable monetary incentive for attackers.

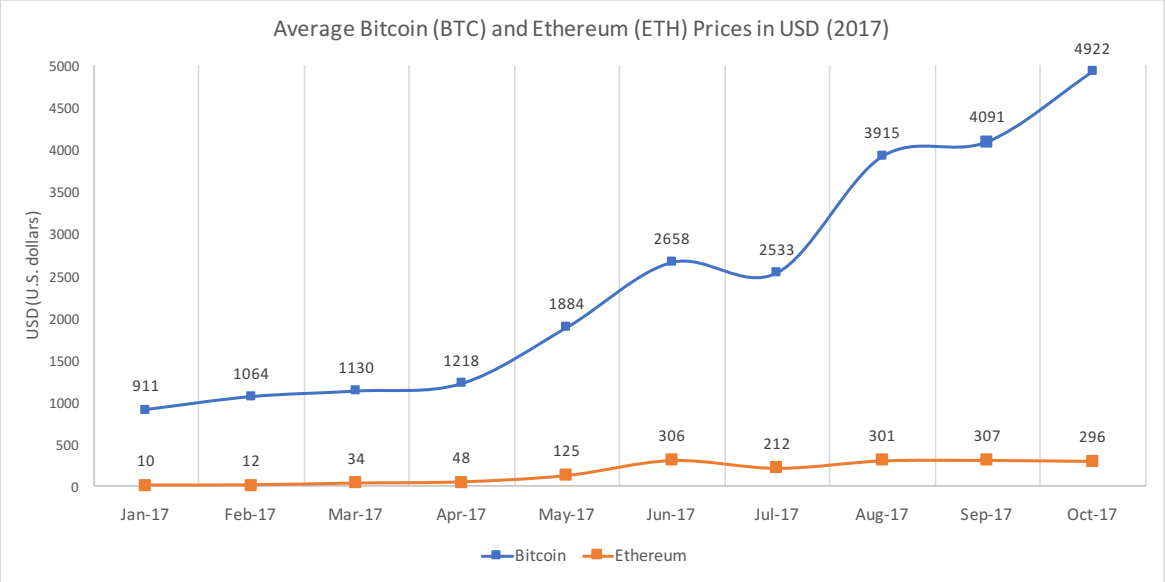


Figure 4.3: Average Bitcoin and Ethereum Prices in USD (2017) [45, 46].

4.2.1 Security vulnerabilities in the blockchain

The following security vulnerabilities have been studied in regards to blockchain security:

- *Wallets attacks.* On Bitcoin, addresses containing bitcoins are stored in wallets. Furthermore, wallets enclose private keys. Any system or user with knowledge of the secret key can claim the bitcoins assigned to the respective account. Currently, several types of wallets exist, including:

- Desktop wallets. Require the installation of software on a personal computer. The official Bitcoin wallet "Bitcoin core" [14] is a desktop wallet, one of the downsides is having to download the entire blockchain which takes disk space and resources.
- Simplified Payment Verification wallets (SPV). This type of wallet does not need to download the entire blockchain. Mobile wallets are usually based on this system.
- Online wallets. Wallets are stored online where access is granted through a third party. One advantage is convenience since users can access the service anywhere [44]. However, since most of these services are centralized, many of the benefits of the blockchain are disregarded. Users should trust a third party for a proper management of balances and private keys. Coinbase [43] is a famous example of an online wallet.
- Hardware wallets. This type of wallet stores private keys in a separate hardware device, similar to a a pendrive for example. One advantage is that this device is maintained offline (usually denominated cold storage) offering protection to attacks that can take place in online wallets.
- Paper wallets. The main advantage of this wallets is that they are cold storage, so the private keys are not online accessible. Additionally, they don't require hardware to function. Therefore, they are not prone to hardware malfunction.
- Bitcoin banks. These type of wallets is based on centralization and trust of a good handle of private keys to a third party.

As per stated by Kaushal, Bagga and Sobti SPV wallets and centralized services such as Bitcoin banks are prone to several types of attacks like: "chain high jacking", "unintentional and intentional transaction suppression", "rewriting chain" and "bait and switch" [110]. Therefore, other alternatives such as hardware, paper and desktop wallets offer more security, but they are not as convenient as SPV wallets, Bitcoin banks and exchange services. However, if the wallets are not encrypted a successful attacker could spend the bitcoins contained on the wallet. Brute force attacks are also a possibility, although it is not likely since attackers would have to break the underlying hash algorithms (SHA-256 in Bitcoin). The Large Bitcoin Collider is a project that seeks to break Bitcoin wallets by performing joint brute force attacks [175]. If these attacks start becoming easier, blockchain could migrate to stronger hash algorithms.

- *Double spending.* Although blockchain deals with the double spending since its inception [132], it is still possible, e.g., a Finney attack, where the attacker forms two transactions: the first, makes a transfer to a third party such as a merchant and the second, makes a transfer to the attacker. When the attacker discovers a new block, he can make a purchase and broadcast his block containing the second transaction. In this process, the first transaction is invalidated. Thus, the merchant does not receive the payment [20]. This type of attack can be avoided by waiting for a certain amount of confirmations (new blocks added to the blockchain). Although, Bitcoin was not initially intended for quick payments some business accept it. Karame and Androulaki reviewed the double spending problem of fast payments on Bitcoin, and they proposed a communication method between peers for prevention of this issue [108]. As mentioned earlier, blockchain deals with double spending by making it expensive. Nakamoto explained an attacker would need to build an alternate blockchain

faster than the public blockchain, like in figure 4.4. The probability of being successful decreases as the attacker needs to keep up with the honest nodes blockchain [132].

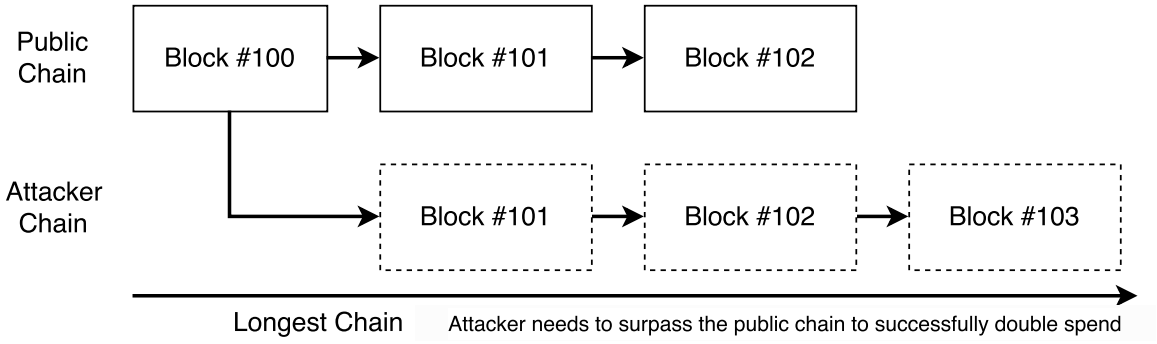


Figure 4.4: Double spend attack on proof-of-work blockchains.

- *Packet sniffing*. An attacker using packet analysis tools could review an user blockchain transactions. Bitcoin Core supports *Tor*, furthermore, providing some level of privacy [15].

- *Selfish mining*. This attack is a variant of a double spend attack but with the intent to waste other nodes resources instead of double spending. This vulnerability consists of a malicious node building a blockchain by its own while the other nodes on the network continue working in the public blockchain. Once the malicious node's private blockchain surpass the public blockchain, the malicious node can broadcast his blockchain [84]. Since in proof-of-work the longest chain is maintained, this causes the honest nodes to waste computing resources, therefore the name of selfish mining.

- *>50% attack or 51 % attack*. This type of attack is probably the most known vulnerability of blockchains based on proof-of-work mechanism. It follows the same procedure as a double spending attack where the attacker builds his own chain trying to surpass the public chain. However, in this case, the attacker possesses 50% or more of the mining power (hash rate) of the network, this means the likelihood of being successful increases due to the proof-of-work protocol nature. To revert a transaction in the past, an attacker needs to rework the proof of work of all the blocks following the transaction's block [117]. To be successful, the attacker would need to invest a lot of resources (depending on the size of the blockchain).

  The consequences of an attack of this nature could be catastrophic for a blockchain since the users would lose trust in the system. Nevertheless, this scenario would not be entirely favorable for the attacker, particularly if an economic goal is pursued. The blockchain currency value along with the usage of the cryptocurrency would likely go down. The result would translate to a less monetary value for double spent transactions, plus the costs of performing the attack and maintaining the majority of the mining power.

- *Timejacking attacks*. This attack consists in altering a node's time counter with the purpose of making the node accept the attacker's chain; this is possible because timestamps are one of the parameters to introduce new blocks [51].

- *Sybil attack*. This vulnerability is specific to peer-to-peer networks; it consists of an attacker creating malicious nodes (identities) with the objective of gaining control of part of the system [60]. A Sybil attack may disconnect users from the blockchain network or try double spending transactions [20].
- *Eclipse attack*. Another peer-to-peer network attack, it consists of setting up malicious nodes around a particular node, controlling the honest node connections and segregating it from the network. This attack can cause the following consequences to the victim: waste computing resources or use them for the attacker's benefit [96].
- *DoS/DDoS*. Peer-to-peer networks don't have a single point of failure opposed to their centralized counterparts. However, Denial of Service and Distributed Denial of Service attacks are still possible in blockchain networks. To exploit this vulnerability, attackers would need to target different nodes on the network, requiring a high number of resources. Blockchains commonly check different parameters that can prevent a denial of service attack, such as the block size, transactions size, the number of signatures and so on [20].

4.2.2 Blockchain Attacks

Although large-scale attacks on Bitcoin's network have not yet occurred, still a potential danger exists due to its increased adoption and value [45]. Attacks on Bitcoin are in general complicated due to its large size. However, smaller blockchains as Krypton have faced 51% attacks; the author was able to double spend and launch DDoS attacks [37].

The most significant attack on Ethereum network so far occurred in 2016. The attack was not against the Ethereum's blockchain itself but to a DAO (Decentralized Autonomous Organization). The attacker exploited code vulnerabilities from the DAO contract and managed to take around 3.6 million ether (equivalent to $50 million US dollars) [164]. Fortunately, due to the contract's constraints, the funds were locked for 28 days, meaning the attacker could not spend them instantly. It the end, a hard fork took place to return the currency to the legitimate owners. Although most of Ethereum's community supported this direction, a group was no convinced about this decision, since it goes against the decentralized nature of the blockchain. Transactions on the blockchain are supposed to be immutable, meaning they are final and cannot change. Due to this debate, the hard fork divided the blockchain in two: Ethereum (supporters of a hard fork) and Ethereum classic (against the hard fork) [53].

A different attack targeting Ethereum's multi-signature wallets took place in 2017. The vulnerability used by the attacker was contained in Parity's wallet's code. The attacker took approx. $32 million US dollars worth of ether. This issue was quickly detected, and the affected wallets were drained to prevent further loss [196].

4.3 Privacy in the blockchain

Many believe Bitcoin transactions are private. However, this is not entirely true since Bitcoin offers pseudo-anonymity through its address based system. If someone can match a Bitcoin address to an identity, then anonymity is lost.

Koshy P., Koshy D. and McDaniel P. [113] analyze Bitcoin traffic with the purpose of mapping Bitcoin addresses to IP addresses. Their results show that it is possible, but with limitations. Transactions sent through Tor are not traceable.

A typical Bitcoin recommendation is to use a different address for every transaction. However, very often this is not the case. In Bitcoin, addresses derive from public keys (see section 2.5).Tools to analysis address reuse are available online [7], these type of mechanisms further help deanonymize Bitcoin transactions.

Some solutions for these privacy concerns have been studied before, including the following:

- *Stealth addresses*. They provide privacy for the recipient of a transaction. The recipient creates a key pair and publishes the corresponding public key, denominated stealth address. Later, the sender can use this public key to create a one-time address, while the recipient generates a secret key from his private key. Therefore, only the recipient can spend the transaction [202].

- *Confidential Transactions*. While Steal addresses seek privacy of the recipient, confidential transactions target the amount of the transaction. Confidential transactions use homomorphic commitments and were generalized by Gregory Maxwell [125].

- *Mixing addresses*. This technique consists of taking many transactions, mix the funds and redistribute. The redistribution may involve performing sub-transactions with smaller amounts and send the operations at different times. All these actions make traceability of transactions a hard task, in particular, if the number of transactions grouped is large. Due to its nature, this technique is often used for illegal activities [202].

- *Ring Signatures*. They allow a member of a group to send a message without the rest of the group knowing who is the actual author, first introduced by Rivest, Shamir and Tauman [156]. Ring signatures only provide privacy for the sender but not for the receiver nor the amount transferred.

- *Zero-knowledge*. Zero-knowledge Proof concept principle is to demonstrate a user knows a secret without revealing it. This type of proof system was first suggested by Goldwasser, Micali, and Rackoff [91]. On blockchain applications, zero knowledge enables privacy of the sender's identity and the transaction's amount.

The paper "A Survey on Security and Privacy Issues of Bitcoin" [50] provides a list of further examples of techniques to improve Bitcoin's privacy.

Think of Bitcoin as a bank account in the cloud, and it's completely decentralized: not the Swiss government, not the American government. It's all the participants in the network enforcing.

Naval Ravikant

5 Application Development

This chapter focus on the development of a decentralized application. Section 5.1 specifies the purpose and requirements of the application. Section 5.2 details the architecture while section 5.3 introduces all the frameworks and technologies used to build this application. Finally, section 5.4 covers the implementation.

5.1 Application Requirements

Human population has increased tremendously in the modern era, causing an escalation of transportation needs. The percentage of people worldwide living in the urban area has changed from 34% in 1960 to over 54% in 2016 [201]. Ride-sharing or carpooling is a term used to denominate the group activity of sharing a ride between two or more people. Ride-sharing provides the following benefits:

- Divide fuel costs.
- Reduce vehicle emissions and promote an environmentally friendly behavior.
- Reduce congested roads.
- Riders avoid driving stress, and both drivers and passengers don't have to spend long commute times by themselves.

Popular solutions have appeared to offer a platform for people to find ride-sharing travels. For example, BlaBlaCar [26] is an online service that promotes long rides between cities. Waze Carpool [191] presents a similar service but through a mobile application. Carpoolworld [38] is a company that offers commutes (more known in the United States), but their services are not only constrained to individuals but also groups (schools, universities, hospitals, etc.). Although Uber [181] possesses carpool services, it is mainly focused on the transportation of individuals from one point to another within a city similar to taxi services.

All these applications are operated by centralized companies which store authentication data (username, passwords) as well as personal information about their users. The purpose of this work is to provide an alternative decentralized version of these platforms.

Some attempts for a similar application have been initiated in the past, including Arcade City [6], a decentralized ride share service like Uber and La'Zooz [116] a platform for users to offer their empty seats in

real time. La'Zooz has a mobile application available for download, currently not working [115]. Unfortunately, it appears neither of these applications have fully materialized and probably were abandoned.

For this reason, "Decentralized Ride Share" seeks to provide a decentralized platform for users to split gas costs using blockchain technology. The functional requirements of Decentralized Ride Share are shown in table 5.1. Depending on the role (rider or driver), specific functionality is either enabled or disabled.

ID	*Functionality*	*Rider*	*Driver*
R1	User should be able to create an account (signup)	X	X
R2	User should be able to update account information	X	X
R3	User should be authenticated (login)	X	X
R4	User should be able to close session (logout)	X	X
R5	User should be able to search for a ride	X	X
R6	User should be able to post a ride offer	X	X
R7	User should be able to book the desired option	X	
R8	User should be able to receive payments		X
R9	User should be able to communicate with the driver (phone, email)	X	
R10	The application should display rides booked by a rider	X	
R11	The application should display rides created by a driver		X

Table 5.1: Application's Functional requirements

5.2 Application Architecture

The application is developed as an Ethereum *dApp* (decentralized application). Ethereum is designed to handle smart contracts, while this is possible in Bitcoin it has its limitations [5]. Ethereum possesses a Turing complete scripting language [200], allowing more complex applications compared to Bitcoin. For this reason, Ethereum platform was chosen for this application.

As presented in section 3.3, a dApp is composed of several components (GUI, a blockchain node, smart contracts, and a blockchain). Figure 5.1 shows this simple structure along with an optional external storage (e.g., IPFS). The decentralized application communicates through a local node to get the current state of the blockchain, while on the other hand accessing static content through an external storage. Additionally, contracts deployed on the blockchain can reference files on the external storage (using for example the hash of the file).

A traditional software application is often based on a client-server architecture, on which a client makes requests to a remote server. A software application usually consists of the following elements [130]:

- *Client*. Composed of two parts: server side, responsible of rendering HTML and a client side, runs scripts and shows the HTML to the user [130].
- *Server*. The web server that receives inquiries from the client and sends the requested data to the client.
  - Presentation Layer. Includes all the components displayed to application's users.

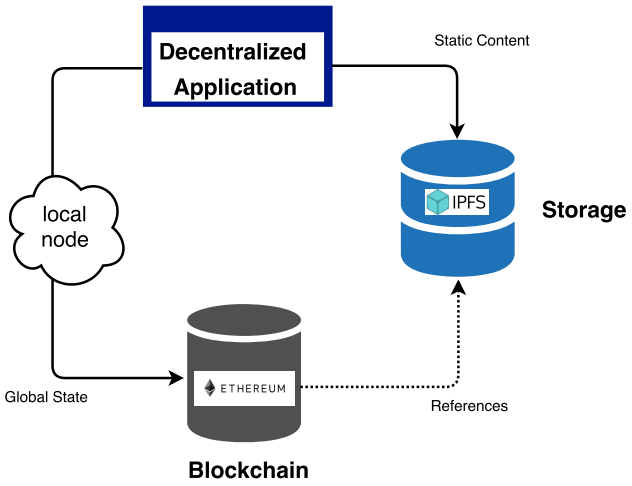


Figure 5.1: Simple structure of a dApp with optional external storage, inspired by [145].

  - Business Layer. Consists of all the application's features and functionality offered to the users.
  - Data Layer. Handles the access to external storage, such as databases.

- *Storage*. A data source, e.g., a database to access an application's data. The server access the data contained in the database.

Nowadays, web applications commonly incorporate a three-tier structure:

- *Front-end*. All the content displayed to the user rendered by the browser. Common front-end content includes: HTML, JavaScript, CSS, etc. It would be the equivalent to the presentation layer of the client-server model.
- *Back-end*. Handles the application's logic and the data access, including databases as well. Common back-end content includes: JavaScript (Node.js), Python, PHP, etc.
- *Middleware*. Represents all the software that allows communication between the front-end and the back-end, often referred as the "plumbing" of an application [199]. Usually, the back-end and middleware are combined and handle together not only the application's logic and database access but also the *APIs* management and further functionality, while the front-end always focus on the presentation of the content.

Decentralized applications have a similar structure: front-end, back-end, middleware. However, the back-end is usually in the form of smart contract. Therefore, a good part of the back-end often moves to the front-end. Table 5.2 provides the application's proposed tools and technologies using this three-tier structure.

Figure 5.2 presents the architecture of the decentralized application. A description of this design is given next:

Front-End	*Back-end*	*Middleware*
JavaScript	Solidity	Truffle
CSS		Web3.js
React		Metamask
JSX		Testrpc
Redux		Other API's and libraries

Table 5.2: Application's Tools and Technologies

- Truffle framework handles the management of contracts. Truffle compiles the Solidity contracts (.sol) using a Solidity compiler. The binaries return to Truffle if no errors are found. This framework helps developers not having to worry about the management of the binaries files.
- Afterwards, the compiled contracts can be deployed to the blockchain (testrpc) using Truffle. After they are deployed, they can be accessed by the dApp.
- Metamask extension runs an Ethereum or a private blockchain node (testrpc). Through this node, Web3.js and Metamask can interact with the blockchain.
- The front-end provides the user interface. Calls to deployed contract take place based on user's input.
- Web3.js can write/read data to deployed contracts by interacting with the local node.
- If a call to one of the deployed contracts changes its state (write), a transaction is required. Transactions are handled by the Metamask extension, through its interface, users confirm transactions. The Metamask extension has access to the user's private key and makes use of it to sign operations.
- Transactions are submitted to the blockchain, and after their confirmation, data is returned to the front-end.

Section 5.3 gives more details of each component on this architecture.

5.3 Frameworks and Technologies

This section introduces the frameworks and technologies used for the development of the decentralized application.

5.3.1 Back-end

Solidity

Solidity is one of the programming languages to build smart contracts on Ethereum blockchain. Although in general is considered a language similar to JavaScript, it has differences, e.g., Solidity is statistically typed [73]. Listing 5.1 presents an example of a simple contract taken from Solidity documentation [70]:

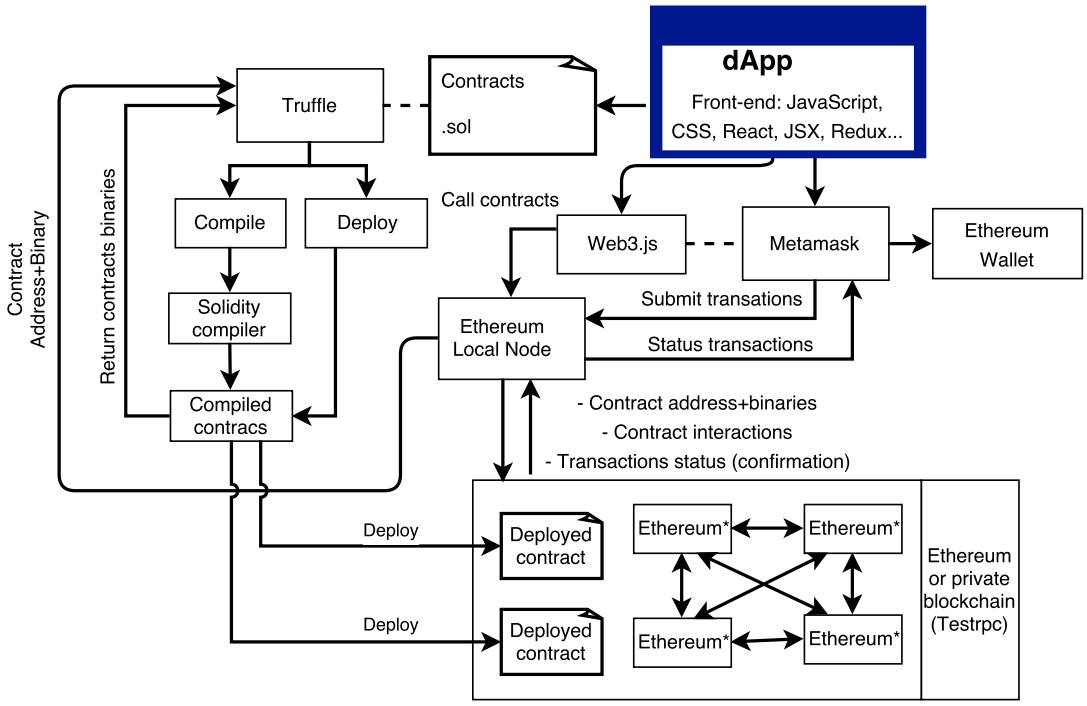


Figure 5.2: Application's Architecture.

```
pragma solidity ^0.4.0;
contract SimpleStorage {
    uint storedData;

    function set(uint x) {
        storedData = x;
    }

    function get() constant returns (uint) {
        return storedData;
    }
}
```

Listing 5.1: Solidity contract example [70]

The first line of the code specifies the version of Solidity employed. Contracts should have a name, in this case, "SimpleStorage". A contract is composed of functions and state variables, in this example, two functions (`set` and `get`') and one variable (`storedData`) of type unsigned integer (uint). The first function `set` will store the value of the received variable `x` (uint type) into the state variable `storedData`. The `get` function will just return the value of `storedData`. This function has a `constant` modifier; this means that the contract status (state) is not affected. Therefore, no transaction takes place. The first function `get` changes the state of the contract since it sets the value of `storedData`.

The omission of semicolons or variable types produces compilation errors. Unfortunately, debugging can become difficult because very often the compiler does not provide specific information about errors. Solidity language was selected for the back-end since it is the most popular language to write smart contracts

on Ethereum. Additionally, it is actively maintained.

5.3.2 Front-end

JavaScript

JavaScript is a programming language commonly employed in web applications. Nowadays, it is considered one of the core technologies of website development, along with HTML and CSS. JavaScript is used for nearly all websites; according to W3Techs (World Wide Web Technology Surveys), almost 95% of all websites use JavaScript [189].

Some of the main characteristics of JavaScript include:

- *Multiparadigm*. JavaScript supports multiple paradigms; a paradigm defines the methodology and style of solving a problem. Some of the paradigms supported by JavaScript include object-oriented, imperative, procedural, functional programming and many others.
- *Weakly typed*. This means the variable type is not reviewed before making an operation, the casting of the variable is done implicitly [183]. JavaScript possesses only a few variable types compared to other programming languages.
- *Interpreted language*. The execution of the code happens without a prior compilation of the code. The interpreter reads the code and executes the requested actions.
- *Prototype-based*. Object orientation is achieved using prototypes in JavaScript. The "objects inherit the property and methods from their prototype" [188]. Due to its popularity and functionality offered (libraries, APIs, etc.) JavaScript is used for the development of the Decentralized Ride Share application.

CSS

CSS stands for Cascade Style Sheets, and as previously mentioned, it is one of the core technologies of modern web development. CSS handles how HTML is shown and presented to the user.

React

React is a JavaScript library for front-end web development. React makes updating and rendering components more comfortable and efficient [85]. React components can be reused and updated as the data changes. Other libraries can be easily incorporated in React.

Other frameworks such as AngularJS [92] or Vue.js [203] could have been used for the development. However, React was selected due to its natural integration with other technologies (Truffle, JSX, Redux).

JSX

JSX allows the conversion of JSX code to JavaScript objects; it has a similar syntax as HTML/XML. JSX enables the insertion of code similar to HTML within JavaScript. This characteristic differs from other libraries or frameworks, on which JavaScript is inserted in HTML code [120]. JSX is often used in combination with React, therefore, integrated into this dApp.

Redux

Redux is a JavaScript library used to handle an application's state [1]. Due to its synergy with React, they are often used together, although Redux can work together with other libraries. Because of its excellent compatibility with React, Redux is likewise adopted in the application.

5.3.3 Middleware

Truffle

Truffle is a framework for building applications on Ethereum. Truffle allows developers to compile, deploy and link contracts [179]. This tool uses migrations to manage and deploy contracts on Ethereum or private networks. One advantage of Truffle is that it provides boilerplates to build dApps, called Truffle boxes. Currently, there are several boxes available that integrate React [178]. The react-auth truffle box [177] is used for the development of "Decentralized Ride Share", it incorporates react-router, redux, webpack and redux-auth wrapper.

There are three commands often used when running Truffle applications:

- `truffle compile`, compiles all the contracts written in Solidity language (.sol extension).
- `truffle migrate`, runs all the new migrations stored in the migrations directory if no new changes are found, then no action takes place.
- `npm run start`, launches the front-end application.

Web3.js

Web3.js is Ethereum's JavaScript *API*; it enables communication between an Ethereum local node and JavaScript files [67]. Web3.js implements a JSON RPC API [71] for communication to occur. Currently, JSON is a very common data exchange protocol. For the developed dApp, Web3.js is the bridge between the front-end (JavaScript) and the blockchain.

Metamask

Metamask is a browser extension that permits interaction with dApps and Ethereum blockchain. Metamask does not require a full Ethereum node to run. This extension injects web3.js API to JavaScript websites, allowing communication with the blockchain. Metamask supports Ethereum blockchain network, test networks (Ropsten, Kovan, and Rinkeby) and Custom RPC (private blockchains) [129].

It is user-friendly since it supports signing blockchain transactions without having to run a separate application. Additionally, it presents balances and other useful data (addresses, transaction information, fees) transparently, making it a good option for running and testing dApps. Although, it is a relatively new technology (and may contain bugs) due to the offered features it is adopted on the dApp.

Testrpc

Testrpc is a platform for testing Ethereum applications. It simulates an Ethereum blockchain, providing the following benefits:

- No mining is required. By default, testrpc creates ten accounts with 100 ether for each account.
- Public and private keys are given for each of the accounts.
- Ethereum transactions usually take between 15-20 seconds, by default transactions block time is "0", so developers wait less for testing.
- The number of initial accounts can be specified according to needs along with other parameters.
- Testrpc is light, meaning the runtime is faster compared to other testing utilities that run a full Ethereum node.

Because of these features, Testrpc can be used for faster development, testing and debugging. Nevertheless, testrpc is a simple tool allowing limited functionality. For a more realistic behavior of a dApp in a blockchain other tools can be used, such as Geth [69]. For the extent of this work, testrpc is sufficient as proof of concept.

Other APIs and libraries

A list of some other libraries, tools, and APIs used to build the dApp are presented next. Many of the libraries or components described can be considered as part of the front-end, due to the functionality they offer.

- *Webpack*. Webpack is a tool that allows the organization (bundle) of JavaScript files easily for their presentation in the browser [192].
- *Google Maps API*. Google's Maps Javascript API [93] permits the user to introduce and select the desired location for traveling.
- *React-places-autocomplete*. This react library incorporates Google's Maps Javascript API for users to select a location in a dropdown form based on their input [99].
- *React Date Picker*. A react component for users to pick dates from a calendar [95].
- *Time Picker*. A react component for users to introduce the desired time in a drop-down form [3].
- *React Table* A react tool to render tables [151], used to present search results and booking options on the application.

5.4 Implementation

This section focus on the implementation of the "Decentralized Ride Share" application.

5.4.1 Back-end

One contract manages the back-end of the application: "RideContract". It is written in Solidity language and can be accessed through Web3.js after it is deployed to a blockchain. The contract offers the application's main functionality and is divided into two parts:

Authentication Part. The purpose of this part of the contract is to handle the authentication process. One data structure of type (`User`) handles the user account information (see listing 5.2). Line 7 creates the data structure `users`, which is `private` meaning other contracts cannot be accessed it. The `address` data type holds a 20-byte value (size of an Ethereum address). For example, `users[pubaddress].name` access a user's account name, the user's public address is used as an identifier inside of this contract.

```
1 struct User {
2     bytes32 name;
3     bytes32 email;
4     uint phone;
5   }
6
7   mapping (address => User) private users;
```

Listing 5.2: User account data structure

In Solidity, `msg.sender` refers to the public address of the user calling a function within a contract. Furthermore, the authentication portion is composed of six functions:

- `signup`. This function (listing 5.3) receives the account information provided by the new user (name, email, phone number). First, it validates the user has provided some information (additional to the front-end review), otherwise it will `revert()`, revert is used to stop the execution of the contract and revert changes done by the function. An alternative to `revert()` is `throw`, however, a disadvantage of throw is that it wastes the gas not used. This revert function is planned to provide additional information about the error in the future (throw is deprecated from Solidity version 0.4.13 onwards)[68]. `Revert()` is preferred since it returns the remaining gas not spent by the function.

  The second part of this function adds the user information to the `users` structure, if there isn't any previous account information.

```
1
2 function signup(bytes32 name, bytes32 email, uint phone) public
    returns (bytes32, bytes32, uint) {
3     // Validates data provided by the user, if no data is provided
    function stops.
4     if (name == 0x0 || email == 0x0 || phone == 0x0)
5     {
6         revert();
```

```
7       }
8
9       // Check if user has an associated account. If yes, returns
        user information otherwise creates new account.
10
11      if (users[msg.sender].name == 0x0 || users[msg.sender].email ==
        0x0 || users[msg.sender].phone == 0x0)
12      {
13          users[msg.sender].name = name;
14          users[msg.sender].email = email;
15          users[msg.sender].phone = phone;
16          regmembers.push(msg.sender);
17
18          return (users[msg.sender].name, users[msg.sender].email,
        users[msg.sender].phone);
19      }
20
21      return (users[msg.sender].name, users[msg.sender].email, users[
        msg.sender].phone);
22  }
```

Listing 5.3: Signup function (inspired by [177])

This function is called by the front-end when a new user wants to register to the application.

- `authenticateuser`. This function authenticates users by starting a new transaction. Users don't have to transfer any ether. However, they have to pay gas fees for the operation. This function is called every time a user wants to access the application.

- `login`. This function receives no parameters; it first checks if the user's account information exists based on the public address, otherwise, the function stops. Afterwards, the user account information is returned. This function is called after the user is authenticated (`authenticateuser`).

- `update`. Users can change their account information. This function provides this functionality by rewriting the `users` data structure. This feature is called through the front-end in the update page.

- `checkmembership`. A function that checks if a user is member of the application. This function is used by the modifier `onlyMembers`. The modifier act as a property of a function, in this particular case, the modifier is used to restrict the access (read/write) to the application's functions (see listing 5.4).

- `getuserinfo`. This function only returns a user's account information, based on the public address provided. It has an `onlyMembers` modifier, so only members can access this function. It is used to obtain a driver's contact information when a user has booked a ride and to get a driver's name when searching for a trip. The underscore of listing 5.4 (line 15) indicates that the function `getuserinfo` code continues; the `checkmembership` function will be executed first and later the code contained on `getuserinfo`.

```
1   function checkmembership () public constant returns (bool) {
2       uint iter;
3
4   // Check if user has permissions to receive account information
    (member of App)
5       for (iter = 0; iter < regmembers.length; iter++) {
6           if (msg.sender == regmembers[iter]) {
7               return true;
8           }
9       }
10      return false;
11  }
12
13  modifier onlyMembers() {
14      if (!checkmembership()) revert();
15      _;
16  }
```

Listing 5.4: "checkmembership" function and "onlyMembers" modifier from RideContract

Ride Management Part. This portion of the contract handles the main features of the dApp: create a new ride, search for a ride and the booking process. It uses three data structures: `rides` holds the specific ride data (id, starting point, destination, date, time, number of seats, cost, etc.), `members` stores the journeys where a user is driving (or has driven before) and the booked rides. The last data structure is `bookings`, contains the booking information (status, reseved seats, total cost). This part of the contract is composed by the following functions:

- `createnewride`. This function creates a new trip as per input from the driver. First, the information is validated, later a `rideID` is generated for the specific ride, and finally, the journey information is added to the `rides` structure. This function gets called after a user submits a create new ride form on the front-end. It requires a blockchain transaction because it changes the state of the contract.

- `createsearchid`. A simple function that increments the `searchid` variable. The purpose is to assign a unique search id for every inquiry performed by users. A `constant` function is used to retrieve the search results. In Solidity, constant functions don't change, therefore, if a user performs the same search twice at different time intervals, the information shown may not reflect up to date information. For example, if Bob (application's user) searches for a ride from Hamburg to Kiel, for the 1st of January of 2018 one week before the trip, he may find two results. One day after, Alice (application's driver) creates a new ride using the same information as Bob inquired before. The result translates to three trips from Hamburg-Kiel for the 1st of January of 2018. If Bob decides to do the same search three days after, he would only find the initial two results. Nevertheless, this is not accurate since Alice's trip is not included in the list. To prevent this type of behavior, a unique search id is created for every search and provided as a parameter when `countresults` is called. This implementation avoids inaccurate search results.

- `getsearchid`. This routine returns the value of the variable `searchid`. It is employed when a user searches for a ride.
- `countresults`. This function iterates through the rides list and counts the total of rides meeting the user's search criteria. Additionally, it stores the ride IDs.
- `checkrideid`. It checks if the given ride ID is valid, used by several functions on this contract.
- `returnride`. This function is called after `countresults`. The specific ride information (from, to, number of seats, date, time, seats) of every ride meeting the user's search criteria is returned.
- `returnride2`. This routine provides the remaining ride information (driver, number of seats available, meeting point, cost). This data is not returned on the previous function `returnride` because of a Solidity limitation. If included in previous function, an "Stack too deep, try removing local variables" error would be displayed after contract compilation.
- `checkseatsandcost`. Returns the number of seats available based on the ride ID provided and the total cost of the booking.
- `prebook`. A function that verifies if the ride ID is valid, if enough seats are available and if the driver wants to book their own ride. The booking information is recorded to the contract on this step. This function executes before booking a trip.
- `fallback function`. In Solidity, a fallback function does not have a name, opposed to normal functions. Furthermore, it cannot receive or return any parameters. This type of function is activated only in two cases: in a function call when no other function matches the name provided and when ether is transferred to the contract [66]. As can be seen on listing 5.5, `payable` is specified in the unnamed function; this term enables the contract to receive ether. `Payable` functions can transfer currency to a given address. The bookride function is triggered by the fallback function, following an ether transference to the contract.

```
1    function () public payable {
2        bookride(msg.sender, msg.value);
3    }
```

Listing 5.5: Fallback function

- `bookride`. It manages the booking process. Before crediting the amount of the booking for the driver, this function verifies the balance of the ride, the status of the reservation (paid/not paid), etc. Also, it updates the number of seats available for the trip and updates the state of the booking. The final step is the transference of the booking cost to the driver; it is done at the end to avoid reentrancy attacks (see section 6.2.1).
- `returnbooking`. Returns the booking information (total payment, booked seats, etc.).
- `getcontractaddress`. Simple function that returns the address of the contract to the front end. It is used to transfer the booking fees for each ride.

- `getrides`. This function returns the specific rides a user is going to drive (or has driven) and the journeys booked so far. It gets called when the user selects "My Rides" from the front-end.

Other contracts. There exist two more contracts "Killable" [167] and "Ownable' [168]', they are standard solidity contracts often used on dApps. "Ownable" create a modifier that allows only the owner to call a function."Killable" allows destructing a contract that can only be done by the owner, using the "Ownable" modifier. Therefore, when a contract has the modifier "Killable", only the creator (owner) of the contract can stop it and receive the remaining funds back.

5.4.2 Authentication

Centralized authentication systems store username and passwords on their servers, allowing attackers to gain access to accounts by exploiting security vulnerabilities. A benefit of blockchain technology is that it enables a more user-friendly authentication mechanism, on which users have the control of their credentials. Metamask is a browser extension that allows effortless interaction with Ethereum's blockchain. An end user can import their Ethereum private keys to the extension directly; the keys are encrypted and stored locally (without a transference to a third-party server).

Authentication is based on public key cryptography; the process works as follows:

1. User imports a private key to Metamask extension.
2. User access the dApp and initiates the login process.
3. Web3.js access user's public key contained in the Metamask provider.
4. A Metamask transaction window appears where the user confirms the transaction.
5. The private key is used to sign the transaction.
6. The transaction is sent to the blockchain where miners process the transaction. If the transaction is confirmed, this means the user's identity is valid, therefore, authenticating the user.
7. The system grants access to the user.

Figure 5.3 describes the authentication process.

Users don't have to provide or remember usernames and passwords. The possession of their keys proves their identity to the dApp.

A different approach, is to perform a "raw transaction". A raw transaction is not submitted to the blockchain for its validation; it happens "offline". It is composed of the next actions:

1. Follow the steps 1 and 2 from the previous method.
2. Web3.js calls a raw (offline) transaction. A hashed message (sha3) is signed used the private key (contained in Metamask).
3. From the signed message a public key is deducted using cryptography tools (ethutil [78]).

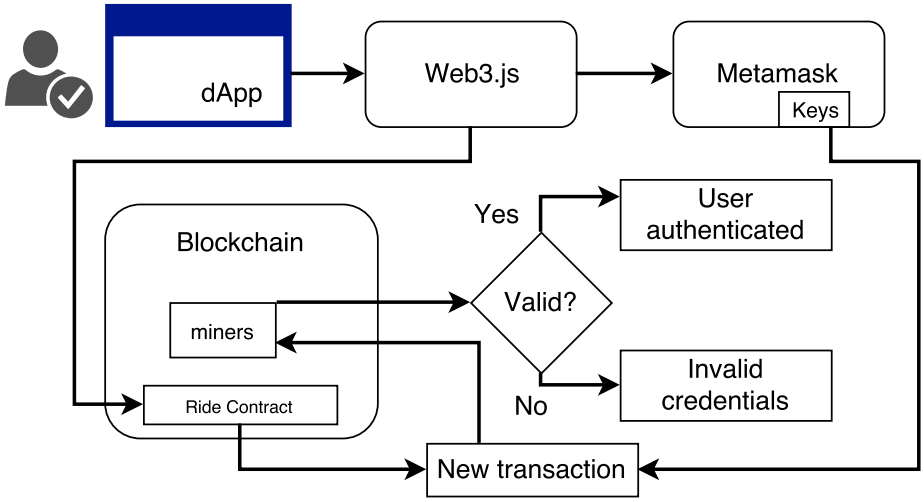


Figure 5.3: Authentication process.

4. The public key (address) deducted from the signed message is compared to the public key of the user.

5. If they match, the dApp authenticates the user.

This method offers the benefit of proving a user's identity without having to generate a blockchain transaction. However, the process to sign raw transactions is vulnerable to ciphertext attacks [87] and to approve malicious transactions.

This method does not provide detailed information about the requester contract or intuitive information about the message signed (figure 5.4). Even though this is a helpful method to authenticate a user, it is not secure from the user perspective. Therefore, the first method was chosen for authentication. There is even a security notice displayed if this technique is used (figure 5.4). There exist additional functions [87] which follow a similar approach but apparently without the security concerns. Unfortunately, they are not currently compatible with the implementation of this dApp.

5.4.3 Front-end

The front-end uses the technologies described before in section 5.3.2. The react-auth Truffle box [177] was used as starting point. The primary function of these tools is presented next:

- *Redux*. Handles the application's state: Web3.js initialization, user's status (authenticated, not authenticated). To do so, Redux implements a store, where actions can be dispatched [152].

- *React*. React is used for the creation of components which are rendered and updated when data is modified. React-router is used for the navigation through the application and works well with Redux; it is used along with react-router-redux.

- *Wrappers*. Redux-auth-wrapper is used to handle the user's state; whether or not it is authenticated. Additionally, two other wrappers control the state of the content visible to authenticated and not authenticated users.

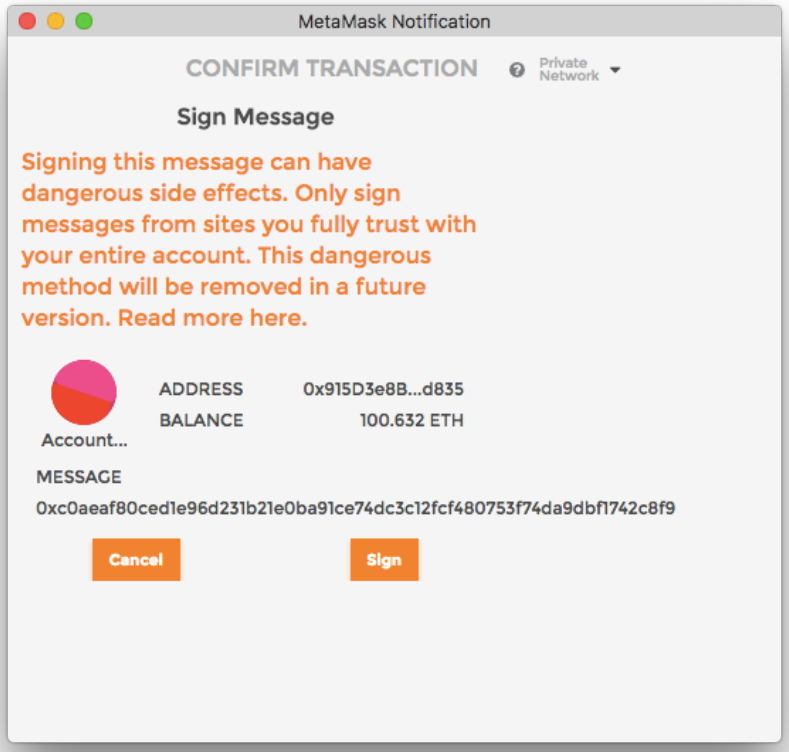


Figure 5.4: Raw Transaction Metamask.

The core structure of the front-end is composed of the following elements:

- *CSS/fonts.* CSS and font files that describe the style of the page. `App.csss` establishes the position and styling of containers, menus and navigation bar of the front-end. They come as default as part of the Truffle box.
- *Layouts.* Contains the main layouts of the application:
  - Home. The home page, starting page where users are presented the dApp.
  - Dashboard. The main page, shown after user authentication. It offers access to the application features: search, create, and view rides.
- *User.* Contains all the components (React) that render the user interface (UI) to the users.
  - layouts. Contains the main components presented to the user. Furthermore, these elements render subcomponents that enable specific functionality.
    - *search.* The main component presented when a user searches for a ride.
    - *create.* The main component showed once the user selects to create a new ride.

    * *booking*. The main component presented after the user inquiries a ride.
    * *signup*. The main component showed when a new user signups. Come as default as part of Truffle box.
    * *profile* The main component presented to users when they decide to update their account information. Come as default as part of Truffle box.
    * *myrides* The main component displayed when users select to view their rides.

  - ui. Contains all the specific components that enable the functionality of the application. Every functionality as presented in the user > layouts is adopted here, plus login and logout features.
  - userReducer. The redux section that allows knowing the user status: login or logout. Come as default as part of Truffle box.

- *util*. Contains the web3.js and reducer interface to know if web3.js is initialized. Come as default as part of Truffle box.
- *other files*:

  - wrappers. The authentication and router wrappers (redux), if the user is not authenticated, redirects the user to the main page for signup. These components enable to show specific menus and information depending on the user's status (authenticated or not authenticated). Come as default as part of Truffle box.
  - App.js. Specifies what is displayed to user's based on their status: not authenticated users (guests) can see the signup, login and home page. By contrast, authenticated users can view the dashboard ("start page"), the profile page and are allowed to logout (See figure 5.5). Come as default as part of Truffle box.
  - index.js. Contains all the routes of the application (redux). Based on the route path, it redirects to the appropriate component, e.g., `/search` redirects the user to the layout component "search" (user > layouts > search).
  - reducer.js. A reducer file the specifies the subcomponents that handle the application's state (routing, userReducer and web3Reducer). Come as default as part of Truffle box.
  - store.js. Creates the redux store using reducer combined subcomponents. Come as default as part of Truffle box.
  - index.html. The index HTML file of the application is contained in the public folder.
  - images. All the images used in the application are contained in the images folder.

Modules

Different modules are incorporated, to offer the features in this application. If the user is authenticated, certain routes are accessible as shown in figure 5.5, including: Welcome page (Home), the Profile page, Dashboard/Start page and logout option. The start page then redirects to other modules depending on user selection (search, booking, create or view rides). If the user is not authenticated, only the welcome page (Home), Signup page and login option are available. A description of every module is presented next:

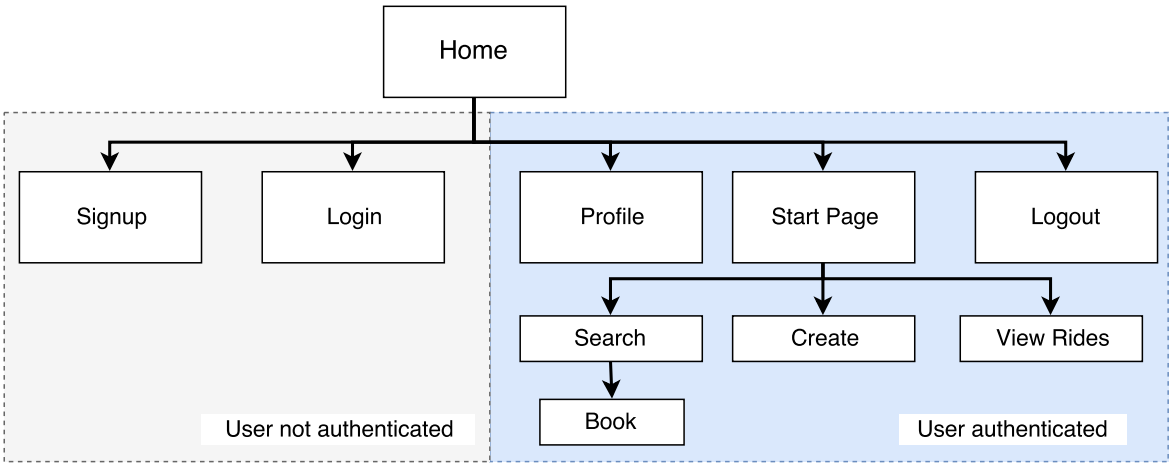


Figure 5.5: Application's Modules/Views.

- *Signup*. The user is presented a Signup form requesting a name, email and phone number. After the fields entered are validated, a `signUpUser` function is called. The `signUpUser` function performs the following actions (figure 5.6):

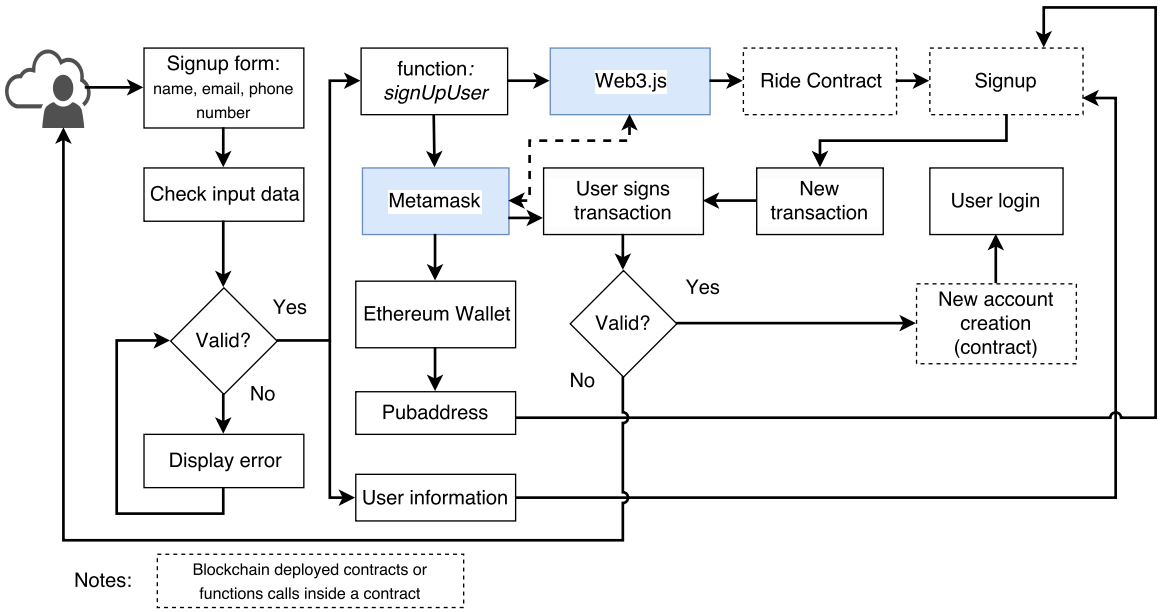
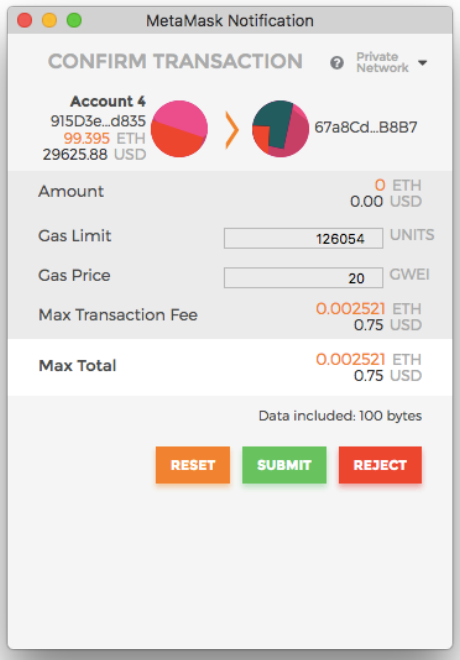


Figure 5.6: Application's Signup process.

1. Creates an instance of the "RideContract".
2. Connects to Metamask provider.
3. Web3.js gets the Ethereum wallet information (public address) contained in the Metamask.
4. Web3.js access the deployed contract and calls the `signup` function.
5. A new account will be created in the back-end after the user confirms a transaction using Metamask. A transaction confirmation window is presented in figure 5.7.
6. Afterwards, the user will be asked to confirm another transaction to access the application (login).



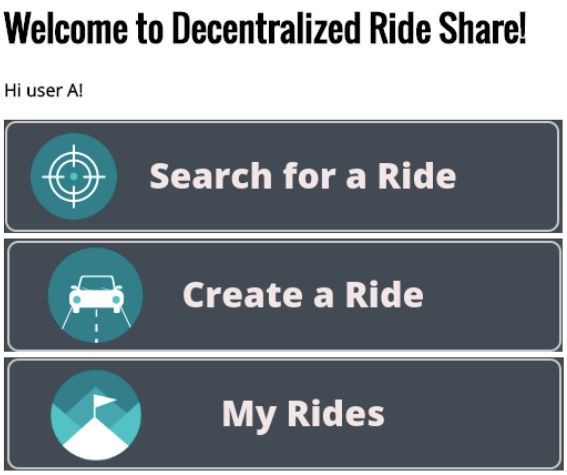


Figure 5.8: Application's Dashboard.

Figure 5.7: Metamask Transaction Confirmation.

- *Profile*. A similar form as the signup form is presented, asking for the same user information. After the fields are completed and validated, an `updateUser` function is called. The steps 1-3 of `signUpUser` function are repeated. Afterwards, the function `update` is called, providing the user information as parameters. Metamask will request the user confirmation of a new transaction (this transaction changes the state of the contract), after successful verification the user information will be updated in the contract.

- *Login*. This module handles the user login which makes use of blockchain technology (see section 5.4.2). After the user selects login, this module performs the same actions as steps 1-3 of signup, later calls the function `authenticateuser`, followed by the `login` function. The purpose of the first

function is to verify the identity of the user; the second function returns the user account information (name, email, phone). Metamask wiil request a user confirmation after calling `authenticate-user`. If everything succeeds, the user is routed to the dashboard page, otherwise back to the signup page, figure 5.9.

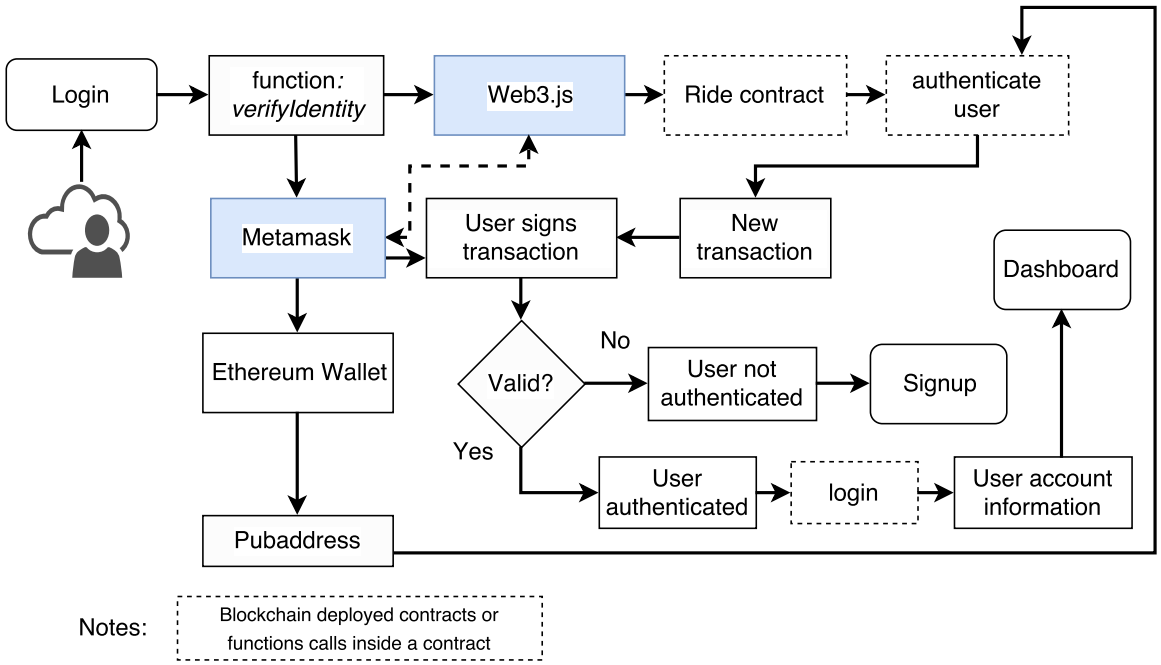


Figure 5.9: Application's Login process.

- *Logout*. The logout function is included as part of the Truffle box. After the user clicks logout, the state of the user changes to logout (redux), causing the user to be "not authenticated". Therefore, the views presented to the user change accordingly. Afterwards, the user is routed Home where signup can take place again.

- *Create Ride*. This option is available on the Dashboard (see figure 5.8). The driver fills a form with detailed information about the ride: starting point, destination, number of seats available, gas retribution, date and time. For the selection of the starting point and destination, Google's Map Javascript API [93] and react-places-autocomplete [99] library are integrated. For the date and time selection, React Date Picker [95] and Time Picker [3] are used respectively.
  After the user submits the form, the fields are verified and if valid a call to the function `createnewride` of the "RideContract" occurs. A Metamask window will show next, and after confirmation of the transaction, the data filled by the driver is written to the contract. (see section 5.4.1). If the operation is valid, the user is redirected to the dashboard, otherwise to the signup page.

- *Search Ride*. In a similar manner as the create a new ride feature, a smaller form requesting a starting point, destination and date is presented (using the same tools as described before). Once the user submits the form and its verified the `searchRide` function is executed, running the following actions:

1. Creates an instance of the "RideContract".
2. Connects to the Metamask provider.
3. Web3.js gets the Ethereum wallet information (public address).
4. A Metamask transaction is required to perform the inquiry.
5. Access the deployed contract and calls the following functions asynchronously, one after another: `createsearchid`, `getsearchid`, `countresults`, `returnride`, `returnride2`, `getuserinfo`. To achieve it, promises are used (see promises). The purpose of these functions calls is to retrieve the ride information according to the parameters specified by the user (rider).
6. After this information is retrieved, the user is routed to the results page, where the data is shown in a table (developed using React Table [151]).

- *Booking*. The search and book processes take place sequentially. A small form will show below the results table; here the user can select one option (if multiple) to book and the desired number of seats. Once this information is validated, the function `bookRide` will run. This function performs the following activities:

  1. Checks the number of seats available (through `checkseatsandcost` contract function).
  2. Starts the booking process. A Metamask window appears to confirm the transaction (through `prebook` contract function).
  3. The user transfers the booking fees to the contract. A second Metamask windows is used to confirm the transaction.
  4. The state of the contract is updated based on the new booking (through `bookride` contract function).
  5. The booking fees are transferred to the driver.
  6. Finally, the user is redirected to the dashboard.

- *View Rides*. This feature allows users to see they ride information. The first section displays the rides a driver is doing in the future or already did. The second portion shows the rides booked, reservation details and the driver contact information (phone number and email address). The following contract functions are called by the `getMyRides` function to obtain the rides the specific user is driving (or has driven before): `getrides`, `returnride` and `returnride2`. The next contract functions are called to get the journeys a user has booked: `returnbooking`, `returnride`, `returnride2` and `getuserinfo`.

Appendix A presents several figures with the user interface of the dApp.

Promises

Promises are used in JavaScript to make asynchronous calls; they can either succeed or fail. Although it is common to use callbacks or nested callbacks, the reading is sometimes not clear, especially if several asynchronous operations take place one after another. Promises help to make this easier: to chain a callback the "`then`" function returns a new promise that gets executed after the previous one is completed [127].

This functionality is implemented in the application when several asynchronous functions are called one after another, e.g., when searching for a ride (listing 5.6).

```
createsearchid()
    .then(getsearchid)
    .then(countresults)
    .then(returnride)
    .then(redirection)
    .catch(function(error){
    ...
```

Listing 5.6: Example of chained promises (searching for a ride)

5.4.4 Middleware

Truffle and Contract Deployment

As mentioned previously, Truffle manages the application's contracts. After Truffle compiles the contracts through a Solidity compiler, Truffle deploys them through migrations. Migrations keep track of the status of the state of the contracts, e.g., if a contract was updated and compiled, the migration status of it changes as well. New contracts need to be added to the "deploy contracts" JavaScript file for their deployment. Truffle keeps the contracts binary files in a separate folder, no further action is required by developers.

Metamask

Metamask has the responsibility of storing the Ethereum's or private blockchain's keys. The user's private key is used to sign transactions and authenticate the user. In this dApp, a Custom RPC is enabled (localhost:8545), this port is used by testrpc. Therefore, Metamask can interact with testrpc private blockchain.

Testrpc

Testrpc simulates Ethereum's blockchain, by default is set to the port 8545. It's a simple tool, but for this reason, testing a contract is faster compared to other means (e.g., Geth). Testrpc provides initial accounts with 100 ether.

Just because something doesn't do what you planned it to do doesn't mean it's useless.

Thomas Edison

6 Results

This chapter provides the results obtained in this thesis. Section 6.1 gives an overall evaluation of the application regarding functionality. Section 6.2 focus on the security and privacy of the application.

Chapter 1 presented the four goals of this thesis:

1. *Review blockchain technology, current state, and applications based on it.*
2. *Give a general analysis of blockchain technology regarding security and privacy.*
3. *Build a decentralized application which uses the blockchain for user authentication.The application should provide a service: a rideshare platform. Users should be able to create, search and book a ride through the application.*
4. *Evaluate the offered service in terms of security and privacy.*

The second and third chapters aimed to cover the first goal of this work. Chapter 2 gave an introduction to blockchain technology and its current status. Chapter 3 provided an overview of blockchain applications.

Chapter 4 focused on security and privacy of the blockchain to cover the second goal of this document. Chapter 5 presented the decentralized application which along with the following section 6.1 will cover the third goal of this thesis.

Finally, to fulfill the last goal, section 6.2 will deal with the security and privacy aspects of the application.

6.1 Functional Evaluation

Section 5.1 presented the functional requirements of the application. Several test cases took place to evaluate if the application meets the specified requirements.

Two user accounts were created through testrpc. Table 6.1 presents the details of the test accounts.

The first test case for functional requirements R1 and R3 is presented in table 6.2. The remaining seven test cases are shown in Appendix B. Since users interact with both the web application and Metamask extension, manual tests took place to perform each test case.

The results of all the test cases are presented in table 6.3.

Test account	*Account (pubaddress)*	*Private key*	*Ether*
Account A	0x915d3e8b...	5755bcd825...	100
Account B	0x983a2756...	45016cb789...	100

Table 6.1: Test accounts

Parameter	*Description/Value*
Test ID	T-01
Requirement Tested	R1, R3
Test Description	Step 1: User access web application and selects "Signup" (navigation bar). Step 2: Test account A is used to signup, providing a name, email and phone number. Step 3: User clicks "signup" and confirms Metamask transaction. Step 4: Users confirms the second operation to login to the application.
Success criteria	1. The application should redirect the user to the signup page. 2. The application should check input criteria and after transactions are validated, redirect to "/dashboard".

Table 6.2: Test case 1

All the test cases passed through the testing methodology described before. However, it is important to denote that while running some tests (additional to the test cases presented), testrpc errors may show. These issues are attributed to testrpc, and it appears to be a common problem with this tool [158, 144]. These errors do not show on a specific contract function call, but instead presented randomly. Therefore, this issue is not considered a bug/issue with the dApp. The problems can be fixed by restarting testrpc.

Given the results obtained from the tests performed the application is considered to meet the functional requirements specified in section 5.1.

6.2 Security and Privacy Evaluation

This section focus on the security and privacy aspects of the decentralized application. Section 6.2.1 will perform a security evaluation, composed of the following steps:

1. General security evaluation of the dApp.
2. Backend evaluation using an automated tool.
3. Frontend evaluation using a web security scanner.
4. Wallet evaluation.

Finally, section 6.2.2 provides a privacy evaluation of the application.

Test ID	*Requirements*	*Success Criteria*		
		1	2	3
T-01	R1, R3	Pass	Pass	N/A
T-02	R2	Pass	Pass	N/A
T-03	R4	Pass	Pass	Pass
T-04	R6	Pass	Pass	Pass
T-05	R5	Pass	Pass	N/A
T-06	R7	Pass	Pass	Pass
T-07	R8, R11	Pass	Pass	Pass
T-08	R9, 10	Pass	Pass	Pass

Table 6.3: Test Results

6.2.1 Security Evaluation

Section 4.1.1 introduced the goals of Computer Security: Confidentiality, Integrity, and Availability. Table 6.4 presents a security evaluation of the decentralized application based on CIA, plus Non-repudiation and Authentication.

General Security Overview of the dApp

Security Goal	*Value*	*Evaluation*
Confidentiality	Moderate	The dApp handles confidentiality through the use of permissions. Every contract and transaction in Ethereum is public. Therefore, anyone can access this information. However, the Ride Contract restricts access to its functions to non-members. The current application design requires a blockchain transaction for most features (signup, login, book a ride, etc.). Therefore, a valid Ethereum account is required to access the application's features. Calling a function by a non-member is possible, however, thanks to the `onlyMembers` modifier implementation, this action has no effect (except wasting gas for the transaction). Still, confidentiality could be improved by restricting access to certain functions even further, for example, a member could request the contact information of another member by calling a contract's function directly. Although this scenario is not likely, it is a possibility. Since no usernames or passwords are used, this type of sensitive information is not at risk. Authentication is handled through the public address (Ethereum account) and the user's private key (through signatures).

Integrity	High	Blockchain technology offers a platform on which information cannot be tampered, or more precisely, it is difficult and costly. The blockchain is often defined as immutable. Nevertheless, the possibility of making changes still exists. A transaction on a blockchain can be reverted by reworking the proof-of-work of all the blocks after the transaction occurred, which is difficult and requires high computation power. This feature provides high integrity to applications based on blockchain technology. Therefore, the dApp provided high integrity regarding security.
Availability	Moderate/Low	This decentralized application needs constant access to the blockchain. For this reason, availability on the dApp is directly related to the availability of the underlying blockchain. The security chapter of this thesis reviewed possible network attacks to the blockchain, e.g., DoS/DDoS. Nevertheless, an attacker would need to attack many nodes on the network to disrupt the blockchain. In comparison, attackers can target specific nodes on centralized networks. Scalability is a common issue of blockchain technology, therefore, this problem would need to be solved first for the dApp to handle millions of users (like other centralized services).
Non-repudiation	High	Digital signatures provide non-repudiation since they ensure that a sender cannot deny sending a message. Since digital signatures are used to submit transactions to the blockchain, non-repudiation is achieved. Furthermore, transactions on the blockchain are public and "immutable" (with the restrictions mentioned before). Therefore, the origin of a transaction made through the dApp can be traced back.
Authentication	High	The application makes use of public cryptography for the authentication process. Additionally, every feature requires the generation of a transaction signed by users. Thus, authenticity of users is constantly checked through the mining process of the underlying blockchain.

Table 6.4: Security evaluation in terms of Confidentiality, Integrity, Availability, Non-repudiation and Authentication

Back-end Security

A security and style validation was performed using Solhint [148]. Solhint is a tool that reviews several security aspects as per stated by the ConsenSys Recommendations for Smart Contracts [49], including the following:

- *Reentrancy*. This is a weakness presented in smart contracts; it occurs when a function is called

again before completing a previous function call. Reentrancy can allow the drain of funds from a contract; as in the famous DAO incident [164]. The result of this reentrancy attack was a hard fork in Ethereum's blockchain. It is recommended to perform the transference of ether after updating state variables and making any changes to the contract, for preventing reentrancy.

- *Mark external contracts.* Another good practice is avoiding calls to external contracts. But when required, untrusted externals calls should be noted. The contract from this dApp prevents this issue by not interacting with external contracts (exceptions are "Killable" and "Ownable" which are designed based on "contract security patterns and best practices" [166]). Furthermore, all dApp backend logic is contained within one contract to prevent interactions with other contracts. Initially, two contracts were written: one to handle Authentication and an additional one for ride management. However, due to the safety issues pointed out, the functionality was combined into one contract.
- *Multiple sends.* Prevent the usage of numerous `send` functions in one transaction.
- *Deprecated functions.* It checks for the usage of unsecured and deprecated functions such as `throw`, `sha3`, `suicide`, `call.value`, `tx.origin`.
- *Mark visibility.* State variables and functions can be set to"public", "private", "internal" or "external". However, marking a function or variable as "private" does not mean others cannot see it, it only prevents the interaction with external contracts [72]. Assigning visibility is not a requirement but is considered a good practice.
- *Check-send-result.* The result of `send` function should be checked to handle the failure of this function. `Transfer` performs this review by default.
- *Compiler Version.* Reviews if the compiler version is above 0.4.
- *Complex fallback functions.* Fallback functions should be kept with simple functionality since this function has a limited gas usage.
- *Function names.* Functions and events should have different names.
- *Others.* Avoid the use of inline assembly, `block.blockhash`, low-level calls, etc. [148].

Besides, Solhint makes a styling review, based on the Solidity Style Guide [74]. The results obtained after running the tool against the `RideContract.sol` contract are presented in table 6.5. A warning regarding compiler version was thrown, followed by two types of styling errors. No security errors were found with this tool. A best practice warning was also given; however, this code complexity is necessary for the `bookride` function to ensure a safe booking process.

Front-end Security

OWASP Zed Attack Proxy (ZAP) [141] is a popular open source tool used to review web application vulnerabilities. This framework can perform the following tests:

Type	*Description*	*RuleID*
warning	Compiler version must be fixed	compiler-fixed
Styling error	Definition must be surrounded with ...	two-lines-top-level-separator
Styling error	Line length must be no more than 120 ...	max-line-length
Styling error	Line length must be no more than 120 ...	max-line-length
Styling error	Line length must be no more than 120 ...	max-line-length
Styling error	Line length must be no more than 120 ...	max-line-lengths
Styling error	Line length must be no more than 120 ...	max-line-length
Styling error	Line length must be no more than 120 ...	max-line-length
Best practice	Function has cyclomatic complexity ...	code-complexity
Styling error	Line length must be no more than 120 ...	max-line-length
Styling error	Line length must be no more than 120 ...	max-line-length
Styling error	Line length must be no more than 120 ...	max-line-length

Table 6.5: Results after running a Smart Contract Security and Styling Tool [148]

- *Proxy Interception/ Passive Scanner*. ZAP permits the review of all HTTP requests and responses from a web application. ZAP scans the application using its spider and reviews all pages found.
- *Active Scanner*. ZAP performs attacks to the pages found.
- *Other*. There are addtional types of tests enabled by ZAP: Spider, review URLs from a website and finds the hyperlinks contained on the URLs, WebSockets, listen to Web-sockets messages and displays them and other features [142].

This tool was used to perform a security evaluation of the dApp's front-end, performing the following steps:

1. *Proxy Setup*. Chrome browser (Version 62.0.3202.89) was used for testing. The browser was set up to connect with ZAP through localhost port 8080. Additionally, ZAP's Root CA certificate was imported to the list of allowed certificates following the procedure from OWASP website [140].
2. *Tool initiation*. Launched ZAP application.
3. *Start dApp*. Started the dApp running on localhost port 3000, development version.
4. *Passive scan*. ZAP continuously monitors the browser and displays alerts accordingly. The dApp application was used (login, logout, create a ride, search a ride and book a trip from a different account) while the tool keeps track of the possible vulnerabilities.
5. *Active scan*. Afterwards, an active scan (penetration testing) was performed using the Quick Start test. The standard policy was applied, which includes injection: buffer overflow, CRLF injection, Cross Site Scripting (Persistent and Reflected), format string error, parameter tampering, server-side code injection, SQL injection and others. Server security: remote file inclusion, path transversal.

Table 6.6 presents a summary of the results obtained following the passive and active scans using ZAP. The complete report is included in Appendix D. The authentication mechanism implemented in the dApp differs from typical web applications. Therefore, the results obtained are limited. However, a security analysis of the wallet is presented later in this evaluation.

Alert Number	*Alert Level*	*AlertType*	*Description*	*Corrective Action*
1	Medium (Medium)	X-Frame-Options Header Not Set	X-Frame-Options header is not included in the HTTP response to protect against "ClickJacking" attacks	Helmet [97] middleware used to set up the HTTP headers (X-Frame Option set).
2	Medium (Medium)	Application Error Disclosure	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception	A react warning message showing in the console, was removed using ESLint [106] (included in truffle box) to prevent disclosure of information.
3	Low (Medium)	X-Content-Type-Options Header Missing	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to "nosniff". This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body	Helmet [97] middleware used to set up the HTTP headers (X-Content-Type-Options "nosniff").
4	Low (Medium)	Web Browser XSS Protection Not Enabled	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the "X-XSS-Protection" HTTP response header on the web server	Helmet [97] middleware used to set up the HTTP headers (XSS filter set).
5	Low (Medium)	Cross-Domain JavaScript Source File Inclusion	The page includes one or more script files from a third-party domain	The file included is the Google Maps API [93], it is considered a secure source.

Table 6.6: Security results after running Passive and Active scans using OWASP ZAP [141]

HTTP headers are common parameters on web applications that can be configured to protect against vulnerabilities. Helmet [97] is a library that incorporates different middleware functions to set up HTTP headers (e.g., prevent Clickjacking, prevent sniff of MIME type, prevent reflected cross-site scripting attacks, and others [97]). This library was included to further deal with the alerts resulted from the passive and active

scans.

Based on these results, it can be concluded that the front-end of dApp is susceptible to similar vulnerabilities as traditional web applications. Therefore, the same security principles and best practices should be considered when developing a dApp.

Wallet Security

As mentioned in section 4.2.1, wallets attacks are a typical vulnerability of blockchain applications. This is due to its importance in handling a user's private keys. The term "cold wallet" is commonly used for wallets that don't possess a connection to the internet, while "hot wallets" do. Some examples of cold wallets include hardware and paper wallets (section 4.2.1).

Metamask stores private keys (encrypted) within a user's computer. Since it interacts directly with dApps, it is considered a "hot wallet", a SPV type (does not require downloading the complete blockchain). SPV wallets are prone to the following kind of attacks [110]:

- *Bait and Switch*. This type of attack exploits the source code of the wallet to replace it with new code with the purpose of retrieving a user's assets. Hybrid and centralized SVPs wallets are vulnerable to this type of attack. For example, OzCoin made use of this vulnerability to recover partially stolen bitcoins [35]. Metamask browser extension runs locally. Therefore, this vulnerability is not considered a high risk.

- *Chain High jacking*. False transactions are including in a longer chain than the official blockchain, confirmed by several nodes. SVPs may accept this chain as the authentic. This type of vulnerability could affect Metamask wallet.

- *Unintentional/Intentional Transaction suppression*. Transactions may take some time to be confirmed, this vulnerability may have an adverse effect if a user considers the transaction as it never occurred, and later the transaction takes place. This can occur unintentionally: slow network, small transaction fee (therefore, miners don't add it to their blocks) or intentionally by an attacker. Bitcoin core handles this problem by keeping track of transactions status.

- *Rewriting chain*. This term refers to the >50% attack or 51 % attack. Reworking the previous blocks is expensive but possible if the attackers have enough computing power, it is a typical attack for all types of wallets and a common vulnerability of blockchain applications.

- *Brute force attack*. Targeted attacks to wallets is difficult but possible by breaking the hashing algorithms, e.g., Large Bitcoin Collider [175].

Although wallets can be stolen and have vulnerabilities, in general, they are considered secure if users properly operate them. Keeping substantial assets in cold wallets is a standard best practice for blockchain applications while using hot wallets for handling smaller assets. Metamask is still in constant development (currently beta version). Therefore, it may be prone to security vulnerabilities until a more stable version is officially released.

6.2.2 Privacy Evaluation

All smart contracts in Ethereum blockchain are publicly available, meaning the code and the variables stored within a contract are public domain. Therefore, a contract can be reviewed by any person [202], this is one significant disadvantage regarding privacy of smart contracts. For this reason, privacy of the dApp is considered *very low*. Although confidentiality is controlled at some level within the contract, privacy cannot be achieved due to the nature of Ethereum's smart contracts.

Nevertheless, private smart contracts may become a reality in the future. Ethereum is starting to explore the implementation of private transactions, through the usage a similar system as zkSnark (Zero-knowledge Proof) [47]. Some research has taken place to experiment with privacy in smart contracts. For example, Hawk [112] allows the creation of private contracts by splitting the logic into two parts: a public contract, executed on a blockchain (such as Ethereum) and a private portion, this logic is implemented outside of a public blockchain. However, the privacy of this system is limited, since a "manager" is the only one who has access to the system. Nevertheless, identities can be kept private if the blockchain implements private transactions [202].

Enigma is described as a decentralized marketplace/platform; the functions run in a distributed form (each node executes a different piece). It seeks to provide privacy by its distributed nature; one node does not hold the complete data [207]. This aspect differentiates Enigma from other blockchains. It has no officially launched, however, the first application (Catalyst [206]) based on this protocol is currently in Alpha release.

7 Conclusion

7.1 Summary

This thesis provided an overview of blockchain technology, its current state and the applications that can be developed with it. Chapter 2 introduced blockchain and the principles behind this technology. Chapter 3 dealt with the different applications that can be built with it and some interesting case scenarios for future applications. Additionally, a security and privacy review of blockchain technology was provided in Chapter 4.

A decentralized application (dApp) was developed based on Ethereum ecosystem using smart contracts. The rideshare application makes use of authentication through the blockchain. Furthermore, the dApp employs the blockchain to save the application's data, which has its disadvantages and limitations (see 7.3). Nevertheless, the constant usage of the blockchain provides some security benefits, e.g., user identity is regularly verified, and integrity is preserved.

A functional review of the dApp was presented in Chapter 6, followed by a security and privacy evaluation. The results from the functional test cases demonstrate dApps can provide similar functionality as their centralized counterparts. The security evaluation shows dApps offer moderate Confidentiality and Availability, but on the other hand, bring high Integrity, Non-repudiation and Authentication thanks to the underlying blockchain technology. For this reason, dApps offer several benefits in the security department.

Nevertheless, smart contracts are a new technology, and for this reason, they are prone to bugs and security concerns. Automated tools exist to verify common security issues, and they are encouraged to meet security guidelines. The front-end of the dApp does not differ much from a traditional web application. Therefore, the same security approach must be followed to reduce the likelihood of an attack. Furthermore, the privacy evaluation exposes important privacy concerns in the current implementation of blockchain applications.

7.2 Future work

This section introduces future work and improvements to the Decentralized Ride Share application.

- *Deploy application in a Test network.* The next step for this dApp would be its deployment to a

test network, before moving to Ethereum's blockchain. This action would provide a more realistic scenario of the behavior of the dApp in a public blockchain environment. Test accounts would need to be setup and ether would need to be mined to access the application's features.

- *Communication within Application*. Currently, the application provides a proof of concept of a decentralized rideshare platform. Application's present state allows the riders to interact with the driver by email or phone number. Further functionality could allow the users to communicate inside the dApp.

- *Rating System*. Like in other platforms, this feature would provide users a property to know how reliable a driver or rider is. Blockchain technology makes tampering results a difficult task. Therefore, this functionality would make use of this blockchain's feature. A reputation score would be calculated based on a performance evaluation of the driver/rider. Moreover, user's reviews could be as well incorporated.

- *Add users Profile page*. A profile page could be added to show specific information about the members of the application. Here users would learn more about the drivers/riders before booking a ride. The rating along with user's reviews could be added to the profile page.

- *External Storage*. The current implementation of the application stores all information within the underlying blockchain. This approach is not the most desirable since storage in the blockchain is expensive. OrbitDb [94] was explored as a potential solution to handle the application's data. However, its current implementation didn't provide the desired functionality. Furthermore, the documentation available was insufficient at the time of development. Other alternatives such as BigchainDB [11] seem promising due to the numerous features they incorporate (query, scalability, permissions, etc.). In addition to a distributed database, big files (e.g. profile pictures) could be stored in a decentralized storage solution such as IPFS [147].

- *Offline Authentication*. This method was described in section 5.4.2. Unfortunately, due to security concerns, it was not implemented. Nevertheless, the future release of functions compatible with testrpc could allow the introduction of this feature. Otherwise, other alternatives methodologies may be explored.

- *Ride Verification process*. In the current implementation, the booking process transfers the full cost of the ride to the driver. This verification feature would limit the amount sent to the driver by a 20-50% of the total cost of the ride. The remaining portion would be transferred once the trip takes place. This process aims to ensure the driver shows per the agreement. At the same time, it would assure the rider attends too. Otherwise, the booking payment would be retained by the driver.

  Along with the rating system, this functionality would provide more reliability to the application and reduce the number of malicious users (who would get poor reviews and thus few rides). Furthermore, a tracking option could be included to allow riders/drivers to share their location and solve disputes. For example, a driver not attending the meeting (at the time and place agreed) would be penalized through this feature.

One possible implementation of this verification process is described in figure 7.1. The following method would take place to verify the ride:

1. The booking process takes place as usual: The rider pays a 20-50% fee for booking the ride. The driver receives this payment.
2. The rider(s) and driver meet at the agreed place and after the ride takes place, the driver accesses the dApp through a web browser and selects the verification ride option.
3. A QR code is displayed through the web application and the rider using a mobile client signs the transaction. This operation transfers the remaining cost of the ride to the driver.
4. The review process can now be accessed through the dApp by the rider(s) and driver.
5. The verification process ends.

Uport [121] offers a mobile application which could be used to implement this feature. It allows signing transactions through the mobile App and managing Ethereum identities. Unfortunately, the mobile application currently only supports "ropsten" test network. There is no current support for private networks or even Ethereum main network. For this reason, this approach was not followed any further. However, this could be explored in the future by transitioning to a test network or adopting a different mobile client.

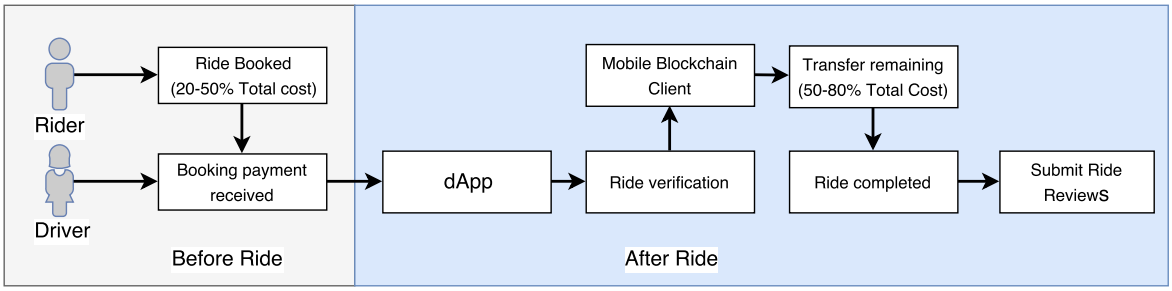


Figure 7.1: Proposed Ride verification process.

7.3 Limitations

The application presents the following limitations on its current implementation:

- Every feature of this application (login, search, create and book a ride) requires signing a blockchain transaction. This aspect makes the overall usage of the application expensive for users. Although this is a disadvantage compared to centralized services, it offers security since the user's identity is continuously verified. A blockchain transaction verification takes around 15 seconds in Ethereum [89], this is a significant constraint in the current design since users would have to wait this time to

interact with the application. However, this aspect could be solved in the future by the incorporation of authentication through an offline method and the transference of the application's data to a distributed network (e.g., IPFS).

- As mentioned in this thesis, blockchains in general face scalability issues. Therefore, running many of the dApp requests at the same time at its current state would generate thousands or millions of transactions pending to process. This would likely slow down and have an adverse effect on the underlying network and the overall performance of the dApp. For this reason, the robustness of the application is limited. Due to this limitation, stress/load testing is not considered part of the scope of this current work.
- The dApp runs on a private blockchain (testrpc). Therefore, it may present a different behavior in a public blockchain. Deployment in a test network is desirable before moving to an open blockchain, such as Ethereum.
- The presented application incorporates many technologies currently in heavy development or in alpha/beta versions (e.g., Metamask, testrpc). Most of the software is provided as it is without warranty, therefore, bugs or compatibly issues can be expected. Appendix C includes a list of the software employed for development and testing of the application.

7.4 Final Remarks

The blockchain is a new technology that has erupted as an alternative to traditional centralized systems. The blockchain has contributed to create a different scheme to exchange assets without the involvement of third parties. Although initially it was limited to currency, the blockchain has evolved and introduced smart contracts and the potential to create decentralized applications, organizations, etc.

Scalability is a known issue of blockchain technology, and this problem should be solved before a broader adoption. A few advancements in blockchain technology are presented next and how could they impact the development of blockchain applications, including the dApp introduced in this work.

Higher transaction throughput. As presented earlier, the current blockchain transaction throughput (e.g., Bitcoin) is far behind compared to other payment systems such as PayPal or Visa (see table 3.2). But current and future network updates are looking to increase the maximum number of transactions supported by a blockchain, e.g., Bitcoin [169, 16] and Ethereum [139, 165].

Reduced cost for transactions. The cost of an operation is relatively high for small transactions (for example, Bitcoin currently between $2.5 and $5 US dollars [21]). However, for more significant operations this is a low fee compared to traditional bank costs, especially for international transactions. Nevertheless, fees may reduce with future updates, e.g., Bitcoin SegWit update [204].

Faster transaction verification. The time between blocks generation dictates how fast a transaction is verified. Therefore, shorter time between blocks translates to a quicker transaction verification process. This feature would enable dApps based on constant interaction with the blockchain to be faster. Systems like Raiden [149] (currently in development) may speed transactions and solve some of the blockchain main scalability issues.

One of the principles of blockchain is based on the power decentralization. A proper distribution of computation power (proof-of-work) or stake (proof-of-stake) through the nodes on the system is highly beneficial to the blockchain. Having a few nodes controlling the network may open the door to security issues, double spent attacks, etc. For this reason, a bigger and more diverse network is good for this technology.

In addition to scalability and technological advancements, blockchain requires a broader adoption of the masses. When Bitcoin first launched, it was synonymous of a platform for illegal activities. Nowadays, this is still a problem but not anymore Bitcoin's unique identifier.With the maturity of blockchain technology and its community growth, many security issues have been solved. However, Blockchain security should not be overlooked due to the assets and valuables it manages. Privacy in Blockchain still has a large room for improvement, but thankfully this is an ongoing research topic.

Despite the challenges and current limitations of the blockchain, its technology has the potential to disrupt and change the way we trade assets, share information, manage organizations, surf the web, identify ourselves, etc.

A

dApp User Interface Figures

The following figures show the user interface where users interact with the dApp.

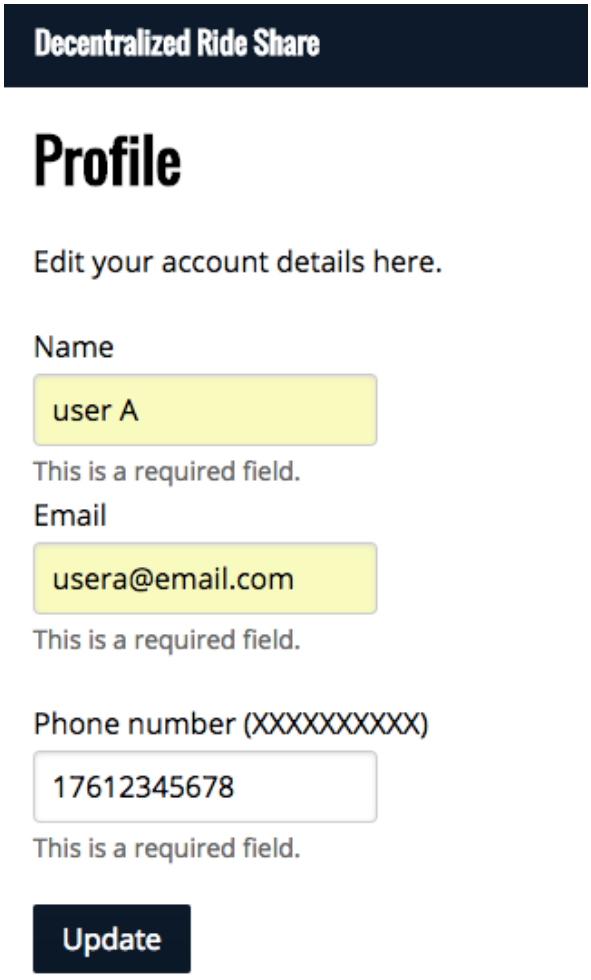


Figure A.1: Profile page of Decentralized Ride Share.

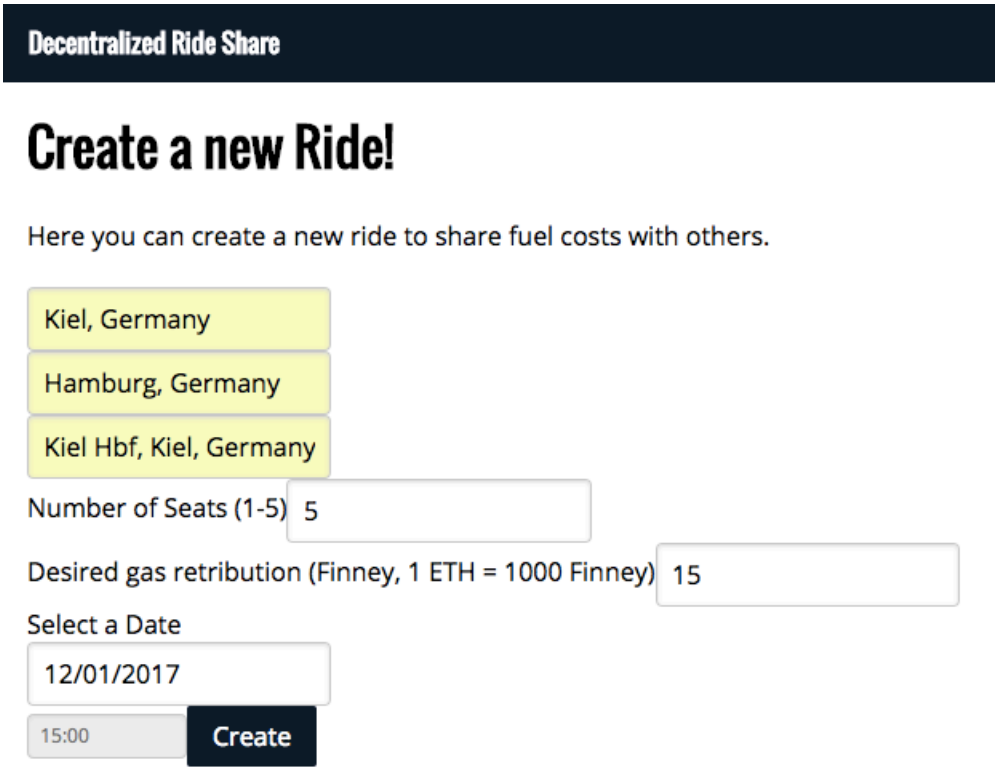


Figure A.2: Create a new ride page of Decentralized Ride Share.

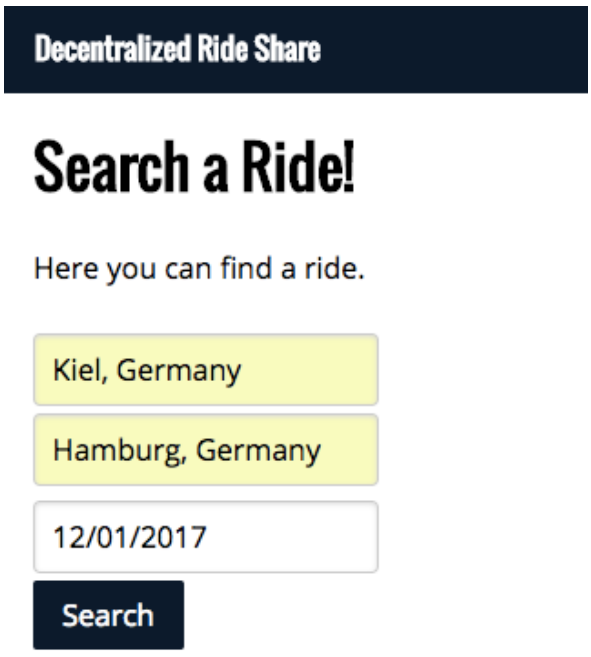


Figure A.3: Search a ride page of Decentralized Ride Share.

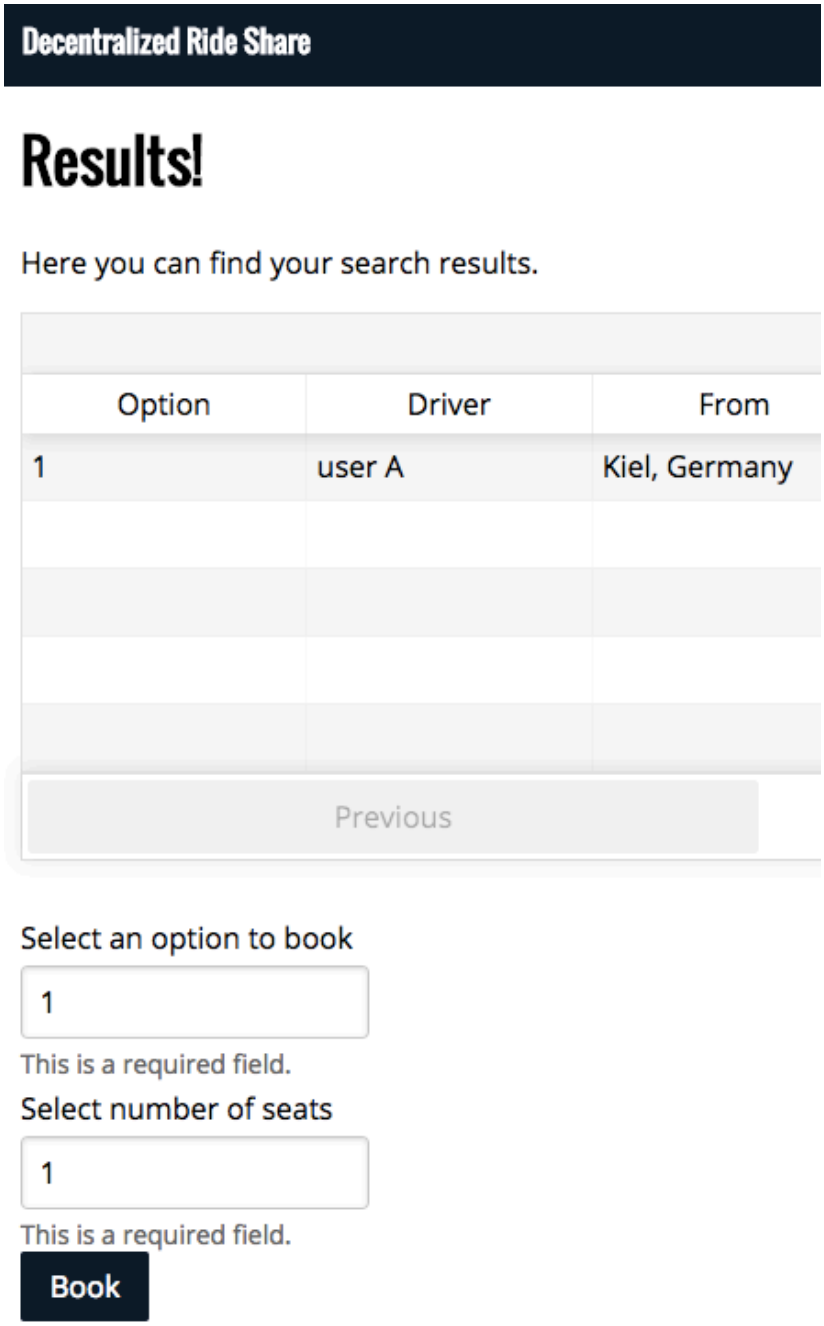


Figure A.4: Results/Booking page of Decentralized Ride Share.

B

dApp Test cases

The remaining test cases are presented in this section.

Parameter	*Description/Value*
Test ID	T-02
Requirement Tested	R2
Test Description	Step 1: User selects “Profile” (navigation bar) from Test account A. Step 2: User updates name, email and phone number. Step 3: User clicks “update” and confirms Metamask transaction.
Success criteria	1. Application should redirect user to profile page. 2. Application should check input criteria and after the transaction is validated, show a message indicating account information was updated.

Table B.1: Test case 2

Parameter	*Description/Value*
Test ID	T-03
Requirement Tested	R4
Test Description	Step 1: User selects “logout” (navigation bar) from Test account A. Step 2: User tries to access “/search page”.
Success criteria	1. The application should logout the user from the application. 2. User should be redirected to the Home page. 3. User should not be able to access search page .

Table B.2: Test case 3

Parameter	*Description/Value*
Test ID	T-04
Requirement Tested	R6
Test Description	Step 1: User selects "Create a Ride" option from the dashboard (Test account A). Step 2: User fills the form and clicks "Create". Step 3: User confirms the Metamask transaction
Success criteria	1. The application should redirect user to create ride page. 2. The application should display a confirmation message. 3. The application should redirect user to the dashboard.

Table B.3: Test case 4

Parameter	*Description/Value*
Test ID	T-05
Requirement Tested	R5
Test Description	Step 1: User selects "Search for a Ride" option from the dashboard (Test account A). Step 2: User inputs a starting point, destination and date and clicks search.
Success criteria	1. The application should redirect the user to search page. 2. The application should display a results table.

Table B.4: Test case 5

Parameter	*Description/Value*
Test ID	T-06
Requirement Tested	R7
Test Description	Step 1: User creates a new account and selects "Search for a Ride" option from the dashboard (Test account B). Step 2: User inputs the previous starting point, destination and date and clicks "Search" (same as Test case 4). Step 3: User fills the booking form and books the ride (confirming the two Metamask transactions).
Success criteria	1. The application should redirect the user to search page. 2. The application should display a results table. 3. The application should redirect to dashboard.

Table B.5: Test case 6

Parameter	*Description/Value*
Test ID	T-07
Requirement Tested	R8, R11
Test Description	Step 1: User selects“My rides” option from the dashboard (Test account A).
Success criteria	1. Application should redirect user to My Rides page. 2. Application should show a booked ride. 3. Account A should be credited the amount of ether of the transaction.

Table B.6: Test case 7

Parameter	*Description/Value*
Test ID	T-08
Requirement Tested	R9, R10
Test Description	Step 1: User selects “My rides” option from the dashboard (Test account B).
Success criteria	1. Application should redirect user to My Rides page. 2. A ride should show under “My rides”. 3. Contact information from the driver (Test Account A) should show.

Table B.7: Test case 8

C

List of Software for Development/Testing

The following is a lit of the software used for the development and testing of the dApp.

Software	*Version*	*Software*	*Version*
Truffle	v3.4.4 (core: 3.4.4)	react-auth-box	v0.1.0
Solidity	v0.4.11 (solc-js)	react-router-redux	v4.0.8
Metamask	v3.12.0	react-datepicker	v0.53.0
Web3.js	v0.18.4	rc-time-picker	v2.4.1
Google Chrome	v62.0.3202.89.100 (64-bit)	react-redux	v5.0.5
macOS Sierra	v10.12.6	redux-auth-wrapper	v1.1.0
TestRPC	v4.0.1 (ganache-core: 1.0.1)	react-table	v6.5.3
react	v15.6.1	react-router	v3.0.5
react-places-autocomplete	v5.3.1	Google Maps API	
OWASP ZAP	v2.6.0	Helmet	3.9.0

Table C.1: List of Software for Development/Testing

Note: Additional dependencies are required for the software described.

D

OWASP ZAP Tool Report

The following pages present the report obtained from running ZAP tool against the dApp.

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	4
Informational	0

Alert Detail

Medium (Medium)	X-Frame-Options Header Not Set
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://localhost:3000
Method	GET
Parameter	X-Frame-Options
URL	http://localhost:3000/
Method	GET
Parameter	X-Frame-Options
Instances	2
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3

Medium (Medium)	Application Error Disclosure
Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	http://localhost:3000/static/js/bundle.js
Method	GET
Evidence	internal error
URL	http://localhost:3000/static/js/bundle.js.map
Method	GET
Evidence	internal error

Instances	2
Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	200
WASC Id	13
Source ID	3

Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://localhost:3000/sockjs-node/info?t=1510658482607
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost:3000/sockjs-node
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost:3000/static/js/bundle.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost:3000/fonts/Oswald-regular.29b3a057ac523422b12afa8f7cf692f7.woff2
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost:3000
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost:3000/
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost:3000/static/js/bundle.js.map
Method	GET
Parameter	X-Content-Type-Options

URL	http://localhost:3000/fonts/Open-Sans-regular.4124088fdd8c315a6d096b65b6cbf428.woff2
Method	GET
Parameter	X-Content-Type-Options
Instances	8
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	**Web Browser XSS Protection Not Enabled**
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	http://localhost:3000
Method	GET
Parameter	X-XSS-Protection
URL	http://localhost:3000/
Method	GET
Parameter	X-XSS-Protection
Instances	2
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it: X-XSS-Protection: 1; mode=block X-XSS-Protection: 1; report=http://www.example.com/xss The following values would disable it: X-XSS-Protection: 0 The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit). Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

	https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14
Source ID	3

Low (Medium)	**Cross-Domain JavaScript Source File Inclusion**
Description	The page includes one or more script files from a third-party domain.
URL	http://localhost:3000
Method	GET
Parameter	https://maps.googleapis.com/maps/api/js?key=AIzaSyA0yokkBr8ka-1JWBnL5b6ZiIuPqMIEnJU&libraries=places
Evidence	<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA0yokkBr8ka-1JWBnL5b6ZiIuPqMIEnJU&libraries=places"> </script>
URL	http://localhost:3000/
Method	GET
Parameter	https://maps.googleapis.com/maps/api/js?key=AIzaSyA0yokkBr8ka-1JWBnL5b6ZiIuPqMIEnJU&libraries=places
Evidence	<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA0yokkBr8ka-1JWBnL5b6ZiIuPqMIEnJU&libraries=places"> </script>
Instances	2
Solution	Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.
Reference	
CWE Id	829
WASC Id	15
Source ID	3

Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://localhost:8545/
Method	POST
Parameter	X-Content-Type-Options
Instances	1
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx

	https://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3

Glossary

API

Application Programming Interface. Allows programmers to adopt a specific functionality by simplifying the required written code, working similarly as a library.. 51, 55

bitcoins

The currency unit on Bitcoin blockchain (BTC).. 8

Block

A blockchain structure unit composed by transactions. A blockchain is formed of a group of blocks.. 4

Collision-resistant

A hash function property were two different inputs cannot generate the same output.. 7

Consensus

The reach of a collective agreement within a group.. 6

dApp

Stands for Decentralized Application, an application running on a peer to peer network without a central server. The term dApp is often used for applications built on Ethereum blockchain.. 30, 49

ether

The currency unit on Ethereum blockchain (ETH).. 8

Hash function

A function that converts an input into a fixed length output.. 7

Miners

A node making use of computing power to validate transactions on a blockchain.. 6

Mining

The verification process performed by miners to validate transactions and blocks.. 6

Nonce

A random number generated with the purpose of obtaining a hash value that meets the requirements of the network.. 7

Public Ledger

A publicly accessible record of the transactions taking place in a network.. 4

Tor

A worldwide network which seeks to provide privacy for internet traffic.. 45

Bibliography

[1] Abramov, D. & Clark, A. (2015). *Redux*. Available at: http://redux.js.org/.

[2] Ali, M., Shea, R., Nelson, J., & Freedman, M. (2017). *Blockstack Technical Whitepaper*. Available at: https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.

[3] Ant Design (2017). *TimePicker*. Available at: https://ant.design/components/time-picker.

[4] Antonopoulos, A. M. (2014). *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st edition.

[5] Araoz, M. (2015). *Smart Contracts and Bitcoin*. Available at: https://medium.com/@maraoz/smart-contracts-and-bitcoin-a5d61011d9b1.

[6] Arcade City, Inc. (2017). *Ridesharing for the people*. Available at: https://arcade.city/.

[7] Atlas, K. (2015). *RFC: Tool for tracking and visualizing Bitcoin address reuse*. Available at: https://www.kristovatlas.com/rfc-tool-for-tracking-and-visualizing-bitcoin-address-reuse/.

[8] Back, A. (2002). *Hashcash - A Denial of Service Counter-Measure*. Available at: http://www.hashcash.org/papers/hashcash.pdf.

[9] Baliga, A. (2017). *Understanding Blockchain Consensus Models*. Available at: https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf?pdf=Understanding-Blockchain-Consensus-Models.

[10] Benantar, M. (2005). *Access Control Systems: Security, Identity Management and Trust Models*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

[11] BigchainDB GmbH (2017). *Meet BigchainDB*. Available at: https://www.bigchaindb.com/.

[12] Bishop, M. (2003). *Computer Security: Art and Science*. Addison-Wesley.

[13] Bitcoin Org (2017). *Bitcoin Developer Guide*. Available at: https://bitcoin.org/en/developer-guide#block-chain.

[14] Bitcoin Project (2017a). *BitcoinCore*. Available at: https://bitcoin.org/en/bitcoin-core/.

[15] Bitcoin Project (2017b). *Excellent Privacy*. Available at: https://bitcoin.org/en/bitcoin-core/features/privacy.

[16] BitcoinCash Org (2017). *BitcoinCash Peer-to-Peer Electronic Cash*. Available at: https://www.bitcoincash.org/.

[17] Bitcoinwiki (2017a). *Address*. Available at: https://en.bitcoin.it/wiki/Address.

[18] Bitcoinwiki (2017b). *Proof of Stake*. Available at: https://en.bitcoin.it/wiki/Proof_of_Stake.

[19] Bitcoinwiki (2017c). *Scalability*. Available at: https://en.bitcoin.it/wiki/Scalability.

[20] Bitcoinwiki (2017d). *Weaknesses*. Available at: https://en.bitcoin.it/wiki/Weaknesses.

[21] BitInfoCharts (2017). *Bitcoin Avg. Transaction Fee historical chart*. Available at: https://bitinfocharts.com/comparison/bitcoin-transactionfees.html#1y.

[22] Bitmain Technologies (2017). *Antminer S9*. Available at: https://shop.bitmain.com/specifications.htm?name=antminer_s9_asic_bitcoin_miner.

[23] BitNation (2017). *BitNation Public Notary (BPN)*. Available at: https://bitnation.co/notary/.

[24] BitShares (2017a). *BitShares - Your share in the Decentralized Exchange*. Available at: https://bitshares.org/.

[25] BitShares (2017b). *Delegated Proof of Stake*. Available at: http://docs.bitshares.org/bitshares/dpos.html/.

[26] BlaBlaCar - Trusted Carpooling (2017). *UberPOOL*. Available at: https://www.blablacar.com/.

[27] BlockAction.io (2017). *Secure and Simplified Solutions for Blockchain Transactions*. Available at: http://blockaction.io/.

[28] Blockchain Info (2017a). *Average Block Size*. Available at: https://blockchain.info/charts/avg-block-size.

[29] Blockchain Info (2017b). *Block #0*. Available at: https://blockchain.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f.

[30] Blockchain Info (2017c). *Hash Rate*. Available at: https://blockchain.info/charts/hash-rate.

[31] Blockchain Info (2017d). *Total Number of Transactions*. Available at: https://blockchain.info/charts/n-transactions-total?timespan=all.

[32] Blockchain Info (2017e). *USD Exchange Trade Volume*. Available at: https://blockchain.info/charts/trade-volume?timespan=alll.

[33] Blum, M., Feldman, P., & Micali, S. (1988). Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88 (pp. 103–112). New York, NY, USA: ACM.

[34] Buntinx, J.P. (2017). *What is Delegated Proof of Stake?* Available at: https://themerkle.com/what-is-delegated-proof-of-stake/.

[35] Buterin, V. (2013). *OzCoin Hacked, Stolen Funds Seized and Returned by StrongCoin*. Available at: https://bitcoinmagazine.com/articles/ozcoin-hacked-stolen-funds-seized-and-returned-by-strongcoin-1366822516/.

[36] Cambridge Dictionary (2017). *Security*. Available at: http://dictionary.cambridge.org/dictionary/english/security.

[37] Campbell, R. (2016). *Krypton Abandons Ethereum for Bitcoin Proof of Stake Blockchain after 51% Attack*. Available at: https://www.cryptocoinsnews.com/krypton-ethereum-bitcoin-proof-of-stake-blockchain-after-51-attack/.

[38] CarpoolWorld (2017). *Carpool*. Available at: https://www.carpoolworld.com/carpool_.html?&form_language=EN.

[39] Carroll, A. & Buchholtz, A. (2014). *Business and Society: Ethics, Sustainability, and Stakeholder Management*, (pp.27). Cengage Learning.

[40] Chaum, D. L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2), 84–90.

[41] Cisco (2017). *Cisco Visual Networking Index: Forecast and Methodology, 2016-2021*. Available at: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf.

[42] Cohen, B. (2008). *The BitTorrent Protocol Specification*. Available at: http://www.bittorrent.org/beps/bep_0003.html.

[43] Coinbase (2017). *Buy and Sell Digital Currency*. Available at: https://www.coinbase.com/home.

[44] Coindesk (2015). *How to Store Your Bitcoins*. Available at: https://www.coindesk.com/information/how-to-store-your-bitcoins/.

[45] Coindesk (2017a). *Bitcoin (USD) Price*. Available at: https://www.coindesk.com/price/.

[46] Coindesk (2017b). *Ethereum (ETH) Price*. Available at: https://www.coindesk.com/ethereum-price/.

[47] Coindesk (2017c). *Ethereum's Byzantium Testnet Just Verified A Private Transaction*. Available at: https://www.coindesk.com/ethereums-byzantium-testnet-just-verified-private-transaction/.

[48] CoinMarketCap (2017). *CryptoCurrency Market Capitalizations*. Available at: https://coinmarketcap.com/currencies/views/all/.

[49] ConsenSys (2017). *Recommendations for Smart Contract Security in Solidity*. Available at: https://consensys.github.io/smart-contract-best-practices/recommendations/.

[50] Conti, M., Kumar, S., Lal, C., & Ruj, S. (2017). A survey on security and privacy issues of bitcoin. *CoRR*, abs/1706.00916.

[51] Corbixgwelt (2011). *Timejacking and Bitcoin*. Available at: http://culubas.blogspot.de/2011/05/timejacking-bitcoin_802.html.

[52] Craib, Richard and Bradway, Geoffrey and Dunn, Xander and Krug, Joey (2017). *Numeraire: A Cryptographic Token for Coordinating Machine Intelligence and Preventing Overfitting*. Available at: https://numer.ai/whitepaper.pdf.

[53] CryptoCompare (2017). *The DAO, The Hack, The Soft Fork and The Hard Fork*. Available at: https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/.

[54] Dean, B. (2017). *Privacy vs. Security*. Available at: https://www.secureworks.com/blog/privacy-vs-security.

[55] Dentacoin (2017). *Dentacoin: An Ethereum-based Token for the Global Dental Industry*. Available at: https://www.dentacoin.com/white-paper/Whitepaper-en1.pdf.

[56] Dickey, M. (2016). *FriendFinder Networks hack reportedly exposed over 412 million accounts*. Available at: https://techcrunch.com/2016/11/13/friendfinder-hack-412-million-accounts-breached/.

[57] Diffie, W. & Hellman, M. (1976). New directions in cryptography. *IEEE Trans. Inform. Theory*, 22, 644–654.

[58] Dingledine, R., Mathewson, N., & Syverson, P. (1890). The right to privacy: the implicit made explicit. In *Harvard Law Review IV*, volume 5 (pp. 193–220).

[59] Dingledine, R., Mathewson, N., & Syverson, P. (2004). Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04 (pp. 21–21). Berkeley, CA, USA: USENIX Association.

[60] Douceur, J. R. (2002). The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01 (pp. 251–260). London, UK, UK: Springer-Verlag.

[61] Dwork, C. & Naor, M. (1993). *Pricing via Processing or Combatting Junk Mail*, (pp. 139–147). Springer Berlin Heidelberg: Berlin, Heidelberg.

[62] EthDoc Org (2017a). *Account Types, Gas, and Transactions*. Available at: http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html.

[63] EthDoc Org (2017b). *Ether*. Available at: http://www.ethdocs.org/en/latest/ether.html.

[64] EthDoc Org (2017c). *What is Ethereum?* Available at: http://ethdocs.org/en/latest/introduction/what-is-ethereum.html.

[65] EtherChain (2017). *Block Number: 0*. Available at: https://www.etherchain.org/block/0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3.

[66] Ethereum (2017a). *Contracts*. Available at: http://solidity.readthedocs.io/en/develop/contracts.html.

[67] Ethereum (2017b). *Ethereum JavaScript API*. Available at: https://github.com/ethereum/web3.js/.

[68] Ethereum (2017c). *Expressions and Control Structures*. Available at: http://solidity.readthedocs.io/en/develop/control-structures.html.

[69] Ethereum (2017d). *Geth*. Available at: https://github.com/ethereum/go-ethereum/wiki/geth.

[70] Ethereum (2017e). *Introduction to Smart Contracts*. Available at: http://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html.

[71] Ethereum (2017f). *JSON RPC API*. Available at: https://github.com/ethereum/wiki/wiki/JSON-RPC/.

[72] Ethereum (2017g). *Security Considerations*. Available at: http://solidity.readthedocs.io/en/develop/security-considerations.html.

[73] Ethereum (2017h). *Solidity*. Available at: https://solidity.readthedocs.io/en/develop//.

[74] Ethereum (2017i). *Style Guide*. Available at: http://solidity.readthedocs.io/en/develop/style-guide.html.

[75] Ethereum Foundation (2017). *Ethereum*. Available at: https://www.ethereum.org/.

[76] Ethereum Github (2017a). *Casper Version 1 Implementation Guide*. Available at: https://github.com/ethereum/research/wiki/Casper-Version-1-Implementation-Guide.

[77] Ethereum Github (2017b). *Proof of Stake Faq*. Available at: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ.

[78] Ethereum Javascript Community (2017). *Ethereumjs-util*. Available at: https://github.com/ethereumjs/ethereumjs-util/blob/master/docs/index.md.

[79] Ethereum Stats (2017). *Ethereum Stats*. Available at: https://ethstats.net/.

[80] Ethereum Team (2017). *Byzantium HF Announcement*. Available at: https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/.

[81] Ethereum Wiki (2016). *Wallet Contract*. Available at: https://github.com/ethereum/mist/wiki/Wallet-Contract.

[82] Etherscan (2017). *Ethereum Average BlockTime Chart*. Available at: https://etherscan.io/chart/blocktime.

[83] Everledger Ltd. (2017). *Everledger*. Available at: https://www.everledger.io/.

[84] Eyal, I. & Sirer, E. G. (2013). Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243.

[85] Facebook Inc. (2013). *React - A Javascript library for building user interfaces*. Available at: https://reactjs.org/.

[86] Finkle, J. & Seetharaman, D. (2014). *Cyber Thieves Took Data On 145 Million eBay Customers By Hacking 3 Corporate Employees*. Available at: https://techcrunch.com/2016/11/13/friendfinder-hack-412-million-accounts-breached/.

[87] Finlay, D. (2017). *The New Secure Way To Sign Data In Your Browser*. Available at: https://medium.com/metamask/the-new-secure-way-to-sign-data-in-your-browser-6af9dd2a1527.

[88] Follow my vote (2017). *Introducing a secure and transparent online solution for the modern age: Follow my vote*. Available at: https://followmyvote.com/.

[89] Foundation, E. (2017). *Ether the crypto-fuel for the Ethereum network*. Available at: https://ethereum.org/ether.

[90] Frost, E. (2017). *Hacking Tesco Bank: The changing nature of bank robbery*. Available at: https://internationalbanker.com/banking/hacking-tesco-bank-changing-nature-bank-robbery/.

[91] Goldwasser, S., Micali, S., & Rackoff, C. (1985). The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85 (pp. 291–304). New York, NY, USA: ACM.

[92] Google (2010). *One framework. Mobile & desktop*. Available at: https://angular.io/.

[93] Google (2017). *Google Maps APIs - Places Library*. Available at: https://developers.google.com/maps/documentation/javascript/places.

[94] Haadcode (2016). *Orbit-db*. Available at: https://github.com/orbitdb/orbit-db#development.

[95] HackerOne Inc. (2016). *React Date Picker*. Available at: https://github.com/Hackerox01/react-datepicker.

[96] Heilman, E., Kendler, A., Zohar, A., & Goldberg, S. (2015). Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)* (pp. 129–144). Washington, D.C.: USENIX Association.

[97] Helmet.js (2017). *Helmet*. Available at: https://github.com/helmetjs/helmet.

[98] Hertig, A. (2017). *How Do Ethereum Smart Contracts Work?* Available at: https://www.coindesk.com/information/ethereum-smart-contracts-work/.

[99] Hibino, K. (2017). *React-places-autocomplete*. Available at: https://github.com/kenny-hibino/react-places-autocomplete.

[100] Hopkins, N. (2017). *Deloitte hit by cyber-attack revealing client's secret emails*. Available at: https://www.theguardian.com/business/2017/sep/25/deloitte-hit-by-cyber-attack-revealing-clients-secret-emails.

[101] Hyperledger (2017). *Hyperledger Whitepaper*. Available at: http://www.the-blockchain.com/docs/Hyperledger%20Whitepaper.pdf.

[102] Inc., H. (2017). *The decentralized, secure, and transparent event management ecosystem*. Available at: https://www.hellosugoi.com/.

[103] International Association of Privacy Professionals (2017). *About the IAPP*. Available at: https://iapp.org/about/what-is-privacy/.

[104] Jakobsson, M. & Juels, A. (1999). Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security*, CMS '99 (pp. 258–272).: Kluwer, B.V.

[105] Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1), 36–63.

[106] JS Foundation (2017). *Configuring ESLint*. Available at: https://eslint.org/docs/user-guide/configuring.

[107] Judmayer, A., Stifter, N., Krombholz, K., & Weippl, E. (2017). Blocks and chains: Introduction to bitcoin, cryptocurrencies, and their consensus mechanisms. *Synthesis Lectures on Information Security, Privacy, and Trust*, 9, 1–123.

[108] Karame, G. O. & Androulaki, E. (2012). *Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin*. Available at: https://eprint.iacr.org/2012/248.pdf.

[109] Katiyar, V., Dutta, K., & Gupta, S. (2010). A survey on elliptic curve cryptography for pervasive computing environment. *International Journal of Computer Applications*, 11(10), 41–46.

[110] Kaushal, P. K., Bagga, A., & Sobti, R. (2017). Evolution of bitcoin and security risk in bitcoin wallets. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)* (pp. 172–177).

[111] King, S. & Nadal, S. (2012). *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. Available at: https://peercoin.net/assets/paper/peercoin-paper.pdf.

[112] Kosba, A., Miller, A., Shi, E., Wen, Z., & Papamanthou, C. (2016). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)* (pp. 839–858).

[113] Koshy, P., Koshy, D., & McDaniel, P. (2014). *An Analysis of Anonymity in Bitcoin Using P2P Network Traffic*, (pp. 469–485). Springer Berlin Heidelberg: Berlin, Heidelberg.

[114] Kript (2017). *Kript - An ecosystem for investing in cryptocurrencies*. Available at: https://drive.google.com/file/d/0Bz8jwqHJrYuQRlBucnBKM09wV1U/view.

[115] L'aZooz (2015). *La'Zooz*. Available at: https://play.google.com/store/apps/details?id=com.lazooz.lbm.

[116] L'aZooz (2017). *A Value system designed for sustainability*. Available at: http://lazooz.org/.

[117] Learn Cryptography (2017). *51% attack*. Available at: https://learncryptography.com/cryptocurrency/51-attack,.

[118] Legatum.io (2017). *Legatum - Protect your legacy*. Available at: http://ec2-107-22-140-84.compute-1.amazonaws.com:3000/#/home.

[119] Levin, L. A. (2003). The tale of one-way functions. *Problems of Information Transmission*, 39(1), 92–103.

[120] Lindley, C. (2017). *What Is JSX?* Available at: https://www.reactenlightenment.com/react-jsx/5.1.html.

[121] Lundkvis, C., Heck, R., Torstensson, J., Mitton, Z., & Sena, M. (2017). *Uport: A Platform for Self-Sovereign Identity*. Available at: https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.

[122] MaidSafe (2017a). *Features of the SAFE Network*. Available at: https://maidsafe.net/features.html.

[123] MaidSafe (2017b). *SAFE Network*. Available at: https://safenetwork.org/.

[124] Malmo, C. (2017). *A Single Bitcoin Transaction Takes Thousands of Times More Energy Than a Credit Card Swipe*. Available at: https://motherboard.vice.com/en_us/article/ypkp3y/bitcoin-is-still-unsustainable.

[125] Maxwell, G. (2016). *Confidential Transaction, the Initial Investigation*. Available at: https://elementsproject.org/elements/confidential-transactions/investigation.html.

[126] McMillan, R. & Knutson, R. (2017). *Yahoo Triples Estimate of Breached Accounts to 3 Billion*. Available at: https://www.wsj.com/articles/yahoo-triples-estimate-of-breached-accounts-to-3-billion-1507062804.

[127] MDN web docs (Mozilla) (2017). *Using promises*. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises.

[128] Merkle, R. C. (1988). *A Digital Signature Based on a Conventional Encryption Function*, (pp. 369–378). Springer Berlin Heidelberg: Berlin, Heidelberg.

[129] Metamask (2017). *Metmask - Brings Ethereum to your browser*. Available at: https://metamask.io/.

[130] Microsoft Patterns (2009). *Microsoft Application Architecture Guide*. Microsoft Press, 2nd edition.

[131] Miessler, D. (2005). *Security: Identification, Authentication, and Authorization*. Available at: https://danielmiessler.com/blog/security-identification-authentication-and-authorization/.

[132] Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Available at: https://bitcoin.org/bitcoin.pdf.

[133] Napster (2017). *Napster -People Powered Music*. Available at: http://us.napster.com/.

[134] NEM (2015). *NEM Technical Reference*. Available at: https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf3.

[135] Nieles, M., Dempsey, K., & Pillitteri, Y. (2017). *An Introduction to Information Security*. Technical report, National Institute of Standards & Technology. Special Publication 800-12 Revision 1.

[136] NIST FIPS PUB 180-4 (2015). *Secure Hash Standard (SHS)*. Standard, *U.S. Department of Commerce*.

[137] NIST Special Publication 800-131A (2011). *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Standard, *U.S. Department of Commerce*.

[138] NxT Community (2014). *NxT Whitepaper*. Available at: https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf.

[139] O'Leary, R. (2017). *Metropolis Today: The Shifting Plans for Ethereum's Next Big Upgrade*. Available at: https://www.coindesk.com/metropolis-today-shifting-plans-ethereums-next-big-upgrade/.

[140] OWASP Project (2017a). *OWASP ZAP 2.6 Getting Started Guide*. Available at: https://github.com/zaproxy/zaproxy/releases/download/2.6.0/ZAPGettingStartedGuide-2.6.pdf.

[141] OWASP Project (2017b). *OWASP Zed Attack Proxy Project*. Available at: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

[142] OWASP Project (2017c). *OWASP Zed Attack Proxy Project*. Available at: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project#tab=Functionality.

[143] PayPal (2017). *PayPal Reports Fourth Quarter and Full Year 2016 Results*. Available at: https://investor.paypal-corp.com/releasedetail.cfm?releaseid=1009339.

[144] PeterBorah- Testrpc Github (2016). *Intermittent "TypeError: Cannot read property 'pop' of undefined"*. Available at: https://github.com/ethereumjs/testrpc/issues/81.

[145] Pirlea, G. (2017). *Lessons learned building a decentralized app*. Available at: https://medium.com/@dranov/lessons-learned-building-a-decentralized-app-d723ab59bfab.

[146] Preneel, B., Bosselaers, A., & Dobbertin, H. (1997). The cryptographic hash function ripemd-160. *CryptoBytes, RSA Laboratories*, 3(2), 9–14.

[147] Protocol Labs (2017). *IPFS is the Distributed Web*. Available at: https://ipfs.io//.

[148] Protofire (2017). *Solhint Project*. Available at: https://protofire.github.io/solhint/.

[149] Raiden (2017). *The Raiden Network*. Available at: https://raiden.network/.

[150] Rainbow Revolution (2017). *Rainbow Revolution*. Available at: http://liquid.qhoss.com/pages/intro.html.

[151] React Tools (2017). *React Table*. Available at: https://github.com/react-tools/react-table.

[152] Redux (2017). *Store*. Available at: http://redux.js.org/docs/api/Store.html.

[153] Ren, L. (2014). *Proof of Stake Velocity: Building the Social Currency of the Digital Age*. Available at: https://www.reddcoin.com/papers/PoSV.pdf.

[154] Republic of Estonia (2017). *E-residency*. Available at: https://e-resident.gov.ee//.

[155] Rivest, R. L., Shamir, A., & Adlemam, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *ACM*, 21, 120–126.

[156] Rivest, R. L., Shamir, A., & Tauman, Y. (2001). *How to Leak a Secret*, (pp. 552–565). Springer Berlin Heidelberg: Berlin, Heidelberg.

[157] Rockwell, M. (2017). *BitCongress - Process For Blockchain Voting and Law*. Available at: http://www.bitcongress.org/BitCongress_Whitepaper.pdf.

[158] rodney757- Testrpc Github (2017). *Tx doesn't have correct nonce error*. Available at: https://github.com/ethereumjs/testrpc/issues/337.

[159] Savers, N. (2015). *Ethash Design Rationale*. Available at: https://github.com/ethereum/wiki/wiki/Ethash-Design-Rationale.

[160] Schuh, F. & Larimer, D. (2017). *Bitshares 2.0: General Overview*. Available at: http://docs.bitshares.org/_downloads/bitshares-general.pdf.

[161] Seeds for Change (2013). *A Consensus Handbook: co-operative decision-making for activists, co-ops and communities*, (pp.6).

[162] ShoCard (2016). *ShoCard - Travel Identity of the Future*. Available at: https://shocard.com/wp-content/uploads/2016/11/travel-identity-of-the-future.pdf.

[163] Sia (2017). *Sia - Your decentralized cloud backend*. Available at: http://sia.tech/.

[164] Siegel, D. (2016). *Understanding The DAO Attack*. Available at: https://www.coindesk.com/understanding-dao-hack-journalists/.

[165] Silva, L. (2017). *Ethereum's Road Map For 2017*. Available at: https://www.ethnews.com/ethereums-road-map-for-2017.

[166] Smart Contract Solutions, Inc. (2016). *Common Contract Security Patterns*. Available at: http://veox-zeppelin-solidity.readthedocs.io/en/latest/contract-security-patterns.html.

[167] Smart Contract Solutions, Inc (2016). *Killable*. Available at: http://veox-zeppelin-solidity.readthedocs.io/en/latest/killable.html.

[168] Smart Contract Solutions, Inc. (2016). *Ownable*. Available at: http://veox-zeppelin-solidity.readthedocs.io/en/latest/ownable.html.

[169] Song, J. (2017). *Understanding Segwit Block Size*. Available at: https://medium.com/@jimmysong/understanding-segwit-block-size-fd901b87c9d4.

[170] Stallings, W. (2011). *Cryptography and Network Security (5th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

[171] State of the dApps (2017). *State Of the dApps*. Available at: https://www.stateofthedapps.com/.

[172] Stockblock (2017). *Stockblock - Artwork Copyrights and attribution on Blockchain*. Available at: http://stockblock.io/StockBlock-whitepaper.pdf?utm_source=website&utm_medium=page&utm_campaign=whitepaper.

[173] Szabo, N. (1996). *Smart Contracts: Building Blocks for Digital Markets*. Available at: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

[174] tbs internet (2017). *All about SHA1, SHA2 and SHA256 hash algorithms*. Available at: https://www.tbs-certificates.co.uk/FAQ/en/sha256.html.

[175] The Large Bitcoin Collider (2017). *About the LBC (FAQ)*. Available at: https://lbc.cryptoguru.org/about.

[176] Trippki (2017). *A Decentralised Ecosystem for Customer Rewards*. Available at: https://trippki.com/assets/docs/TrippkiWhitePaper_Sept17.pdf.

[177] Truffle (2017a). *react-auth*. Available at: http://truffleframework.com/boxes/react-auth.

[178] Truffle (2017b). *Truffle Boxes*. Available at: http://truffleframework.com/boxes/.

[179] Truffle (2017c). *Your Ethereum Swiss Army Knife*. Available at: http://truffleframework.com/.

[180] Trustnodes (2017). *Ethereum May Reduce Mining Reward and Inflation by 40%*. Available at: http://www.trustnodes.com/2017/07/15/ethereum-may-reduce-mining-reward-inflation-40.

[181] Uber Technologies Inc. (2017). *UberPOOL*. Available at: https://www.uber.com/ride/uberpool/.

[182] U.S. Energy Information Administration (2017). *Frequently Asked Questions*. Available at: https://www.eia.gov/tools/faqs/faq.php?id=97&t=3.

[183] Vaggalis, N. (2010). *Weakly Typed Languages*. Available at: http://www.i-programmer.info/programming/theory/1469-type-systems-demystified-part2-weak-vs-strong.html.

[184] Vanhoef, M. & Piessens, F. (2017). *Key Reinstallation Attack: Forcing Nonce Reuse in WPA2*. Available at: https://papers.mathyvanhoef.com/ccs2017.pdf.

[185] Vasin, P. (2014). *BlackCoin's Proof-of-Stake Protocol v2*. Available at: https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf.

[186] Vermeulen, J. (2016). *VisaNet - handling 100,000 transactions per minute*. Available at: https://mybroadband.co.za/news/security/190348-visanet-handling-100000-transactions-per-minute.html.

[187] Vorick, D. & Champine, L. (2014). *Sia: Simple Decentralized Storage*. Available at: http://www.sia.tech/whitepaper.pdf.

[188] W3Schools (2017). *JavaScript Object Prototypes*. Available at: https://www.w3schools.com/js/js_object_prototypes.asp.

[189] W3Techs - World Wide Web Technology Surveys (2017). *Usage of JavaScript for websites*. Available at: https://w3techs.com/technologies/details/cp-javascript/all/all.

[190] Waves Plattform (2016). *WAVES Whitepaper*. Available at: https://blog.wavesplatform.com/waves-whitepaper-164dd6ca6a23.

[191] Waze Mobile (2017). *WazeCarpool*. Available at: https://www.waze.com/carpool/.

[192] Webpack (2017). *Core Concepts*. Available at: https://webpack.js.org/concepts/.

[193] Westin, A. (1967). *Privacy and Freedom*. New York: Atheneum.

[194] Whitman, M. E. & Mattord, H. J. (2011). *Principles of Information Security*. Boston, MA, United States: Course Technology Press, 4th edition.

[195] Wilkinson, S., Boshevski, T., Brandoff, J., Prestwich, J., Hall, G., Gerbes, P., Hutchins, P., & Pollard, C. (2016). *Storj - A Peer-to-Peer Cloud Storage Network*. Available at: https://storj.io/storj.pdf.

[196] Wilmoth, J. (2017). *Hackers Seize $32 Million in Ethereum in Parity Wallet Breach*. Available at: https://www.cryptocoinsnews.com/hackers-seize-32-million-in-parity-wallet-breach/.

[197] Winters, T. (2016). *Uncle Mining in Ethereum*. Available at: https://www.ethnews.com/uncle-mining-in-ethereum.

[198] Wizard, O. I. (2017). *Welcome to ICO Wizard*. Available at: https://wizard.oracles.org/.

[199] Wodehouse, C. (2017). *A Beginner?s Guide to Back-End Development*. Available at: https://www.upwork.com/hiring/development/a-beginners-guide-to-back-end-development/.

[200] Wood, G. (2017). *Ethreum: A secure decentralized generalized transaction ledger* . Available at: https://ethereum.github.io/yellowpaper/paper.pdf.

[201] World Bank Group (2017). *Urban population (% of total)*. Available at: https://data.worldbank.org/indicator/SP.URB.TOTL.IN.ZS.

[202] Yang, D., Gavigan, J., & Wilcox-O'Hearn, Z. (2016). *Survey of Confidentiality and Privacy Preserving Technologies for Blockchains*. Available at: https://z.cash/static/R3_Confidentiality_and_Privacy_Report.pdf.

[203] You, E. (2014). *The Progressive JavaScript Framework*. Available at: https://vuejs.org/.

[204] Young, J. (2017). *Bitcoin Fees Are High, But They Will Decrease With SegWit Soon*. Available at: https://www.cryptocoinsnews.com/bitcoin-fees-high-will-decrease-segwit-soon/.

[205] Zetter, K. (2009). *Bullion and Bandits: The Improbable Rise and Fall of E-Gold*. Available at: https://www.wired.com/2009/06/e-gold/.

[206] Zyskind, G., Kisagun, C., & Fromknecht, C. (2017a). *Enigma Catalyst: A machine-based investing platform and infrastructure for crypto-assets*. Available at: https://token.enigma.co/pdf/Enigma-Catalyst.pdf.

[207] Zyskind, G., Nathan, O., & Pentland, A. (2017b). *Enigma: Decentralized Computation Platform with Guaranteed Privacy*. Available at: https://token.enigma.co/pdf/Enigma-Full.pdf.