NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

Masters Thesis

Accountable and privacy preserving data processing via distributed ledgers

Christos S. Nasikas

ATHENS

MAY 2018

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Διπλωματική Εργασία

Λογοδοτούμενη επεξεργασία δεδομένων που διατηρεί την ιδιωτικότητα μέσω κατανεμημένων μητρώων

Χρήστος Σ. Νασίκας

ΑΘΗΝΑ

ΜΑΙΟΣ 2018

Masters Thesis

Accountable and privacy preserving data processing via distributed ledgers

Christos S. Nasikas
A.M.: M1443

SUPERVISOR: Aggelos Kiayias, Associate Professor EKPA

EXAMINATION COMMITTEE:

Aggelos Kiayias, Associate Professor EKPA

Smaragdakis Yannis, Professor EKPA

MAY 2018

Διπλωματική Εργασία

Λογοδοτούμενη επεξεργασία δεδομένων που διατηρεί την ιδιωτικότητα μέσω κατανεμημένων μητρώων

Χρήστος Σ. Νασίκας
Α.Μ.: Μ1443

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Άγγελος Κιαγιάς, Αναπληρωτής Καθηγητής ΕΚΠΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Άγγελος Κιαγιάς, Αναπληρωτής Καθηγητής ΕΚΠΑ

Σμαραγδάκης Γιάννης, Καθηγητής ΕΚΠΑ

ΜΑΙΟΣ 2018

ABSTRACT

Data are gathered constantly, grow exponentially, and are considered a valuable asset. The need for extensive analysis has emerged by various organizations and researchers. However, they can be sensitive, private, and protected by privacy disclosure acts making data processing by third-parties almost impossible. We propose a protocol for data processing where data controllers can register their datasets and entities can request data processing operations by data processors. A distributed ledger is used as the controller of the system serving as an immutable history log of all actions taken by the participants. The blockchain-based distributed ledger provides data accountability, auditability and provenance tracking. We also use a Zero Knowledge Verifiable Computation scheme where a data processor is enforced to produce a proof of correctness of computation without revealing the dataset itself that the requestor verifies. This records the fact that correct processing has taken place without disclosing any information about the data.

SUBJECT AREA: Blockchain

KEYWORDS: distributed ledger, accountability, privacy preserving, data processing, zero knowledge proofs

ΠΕΡΙΛΗΨΗ

Ο όγκος των δεδομένων που συλλέγονται καθημερινά σημειώνει εκθετική αύξηση, ενώ η κατοχή τους θεωρείται πολύτιμη. Η ανάγκη για εκτένη ανάλυση έχει αναδειχθεί μέσα από το έργο διαφόρων ερευνητών και οργανισμών. Ωστόστο, τα δεδομένα αυτά μπορεί να είναι ευαίσθητα και να υπάγονται σε ρυθμιστικές νομοθεσίες απορρήτου κάνοντας την επεξεργασία από τρίτους αδύνατη. Προτείνουμε ένα πρωτόκολλο στο οποίο επεξεργαστές δεδομένων (data processors) έχουν την δυνατότητα να καταχωρήσουν σύνολα δεδομένων (datasets) για τα οποία μπορούν να γίνουν αιτήσεις επεξεργασίας οι οποίες διεκπαιρεώνονται από επεξεργαστές δεδομένων (data processors). Ένα κατανεμημένο μητρώο (distributed ledger) χρησιμοποιείται ως διαχειριστής του συστήματος λειτουργώντας ως ένα αμμετάβλητο ιστορικό όλων των ενεργειών των συμμετεχόντων. Το κατανεμημένο μητρώο παρέχει τις ιδιότητες της λογοδοσίας, του ελέγχου και της παρακολούθησης της προέλευσης των δεδομένων. Επίσης, χρησιμοποιείται ένα σχήμα Μηδενικής Γνώσης Ορθότητας Υπολογισμού (Zero Knowledge Verifiable Computation) μέσα από το οποίο οι επεξεργαστές δεδομένων υποχρεούνται να παράξουν μια απόδειξη ορθότητας υπολογισμού, χωρίς να αποκαλείψουν το ίδιο το σύνολο δεδομένων, την οποία ο αιτών (data requestor) και επαληθεύει. Κατά αυτό τον τρόπο πιστοποιείται το γεγονός ότι πραγματοποιήθηκε η σωστή επεξεργασία δεδομένων χωρίς να αποκαλυφθούν επιπλέον πληροφορίες σχετικά με αυτά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Blockchain

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: κατανεμημένο μητρώο, υπευθυνότητα, ιδιωτικότητα, επεξεργασία δεδομένων, αποδείξεις μηδενικής γνώσης

ACKNOWLEDGEMENTS

I wish to first and foremost thank my father for his total support through those year of my studies and not only. He was always there at my most harsh moments of my life. His wisdom and inner peace is crucial to my life.

I am indebted to my dear friend George Argyros for his help, mentoring, guidance, support, and for introducing me to the world of security and cryptography. Without him it would be impossible to graduate.

I thank Dionysis Zindros for his key observations, guidance and support. He was always patiently trying to answer my questions.

I would like to thank all the members of the Cryptography Lab at the University of Athens for their support and help during my stay there.

I would like also to thank my love Katerina for her support, patience and understanding during my postgraduate course. She was always there for me.

Finally, I thank my sister, my brother, and my mother for being my dear family. I love you all.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SOURCE CODES

1. INTRODUCTION

1.1 Overview

Some say we live in the Big Data era where data are being created and gathered in a rapid pace [181]. Technological breakthroughs over the past ten years in software and hardware have given rise to unprecedented ability to store and analyze data from various sources – digital or, digitized, wearables or IoT devices with all kinds of sensors – to personal data collected from everyday routine activities, medical records, bank accounts, mobility or CCTV. Such data are commonly used in statistical techniques which can identify patterns and relations that can lead to the predictions of otherwise unknown events. They can, therefore, be beneficial for researchers and organizations, anyone who is interested in making sense of complex behavioral, financial, climatic, anthropological, and chemical phenomena. However, data related to people, personal data, are and should remain sensitive in terms of privacy. Many countries have forced correspond legislation, rendering such data legally inaccessible to anyone. The value of human data is huge, in financial, scientific and political terms. Harvesting and processing data in a privacy preserving manner is both very crucial and very challenging.

In 2008, the emergence of the blockchain technology highlighted new ways of data exchange. Its main property being its ability to achieve consensus among trustless entities connected through a decentralized network [79], the blockchain could provide the cornerstone technology and logic for data sharing without compromising privacy.

The blockchain utilizes the cryptographic primitives necessary for transaction trackability and data provenance tracking. It is an immutable log that keeps record of all the participants's actions. It ensures the accountability of the participants consequently enforcing non-repudiation. Such properties of the blockchain make it a useful tool for data exchange and processing.

Choosing to use the blockchain to perform as a system controller means also having to deal with its limitations. It is argued that the blockchain is unsuitable for high performance transactions [158, 179] or as a database replacement. The existing technology behind blockchain is not made for big data. The amount of data that blockchain can store and process is very limited and off-chain data frameworks needs to be combined.

The implementation of the above concepts, trying to overcome the recurring problems, is the main goal of this thesis. Alongside the previous goals, emphasis is put on zero knowledge verifiable computations which can produce a proof of correctness of computation over a dataset allowing the verification of the proof hiding sensitive input.

1.2 Thesis structure

The present thesis is organized as follows. In Section 2, we give common definitions and basic properties of various cryptographic primitives used in the following sections. In Section 3 the blockchain is presented – its history, its internal components, and how these components work. In Section 4 we analyze the various obstacles of privacy preserving data sharing and how the blockchain could be a useful component in the sharing ecosystem. In Section 5 and 6 a solution is being developed and analyzed, including security assumptions, high level architecture and implementation details. In Section 7, we present

the results of various experiments regarding blockchain data storage and zkSNARK construction. Various ideas for system improvement are presented in Section 8. Further and related work is also discussed in Section 9. Finally, a concluding statement summing up the findings of this research is included in Section 10.

2. PRELIMINARIES

2.1 Overview

Cryptography is the art and science of securing digital information, transactions, and distributed computations in the presence of third parties [101]. Cryptography is directly related to data privacy and anonymity requirements. The blockchain is a good example of cryptography embracement to enable the sought trust needed for the exchange of digital assets. In this section we review the basic building blocks that cryptography provides to a data sharing system when the blockchain is used. In this Section, cryptographic schemes are introduced in order to lay the ground for the following research. As a result, the exposition style will not be formal in well known cases.

2.2 Cryptographic Hash Functions

A *hash function* is a function which takes an input of arbitrary length and returns a fixed-length value [30, 101, 102, 119]. The hash output value is called *digest*.

Hash functions have many applications. They are used in data structures – such as hash tables – in authentication schemes, password verification and data identifiers to name just a few.

A hash function guarantees at a minimum that for the same input yields the same output. The size of the output is fixed, thus the output range is finite. For this reason it is possible two different inputs produce the same output. This phenomenon is called *collision*.

Cryptographic hash functions have much stronger properties than regular hash functions. Ideal cryptographic hash functions should be easily computable, noninvertible and collision-resistant [101, 102, 119]. A cryptographic hash function H is a deterministic polynomial algorithm that takes as input any given string $x \in \{0,1\}^*$ and outputs a string $H(x) \in \{0,1\}^k$ where k is of fixed size. In the case of a hash function H, a collision is a pair of distinct messages m_0, m_1 where $m_0 \neq m_1$ and $H(m_0) = H(m_1)$. A hash function H is collision-resistant if it is infeasible for any polynomial-time algorithm to find collisions.

Three levels of security [101] can be identified:

- **Preimage resistance**: Given a digest h it is hard to find any message m with $H(m) = h$
- **Second preimage resistance**: For any given message m_0 it is hard to find a second message $m_1 \neq m_0$ such as $H(m_0) = H(m_1)$
- **Collision resistance**: It is hard to find a pair of messages m_0, m_1 where $m_0 \neq m_1$ and $H(m_0) = H(m_1)$

Collision resistance is the strongest security property and a requirement for cryptographic hash functions.

In a data sharing system, cryptographic hash functions can be used to provide file integrity verification and provenance tracking [13, 181]. File modifications in transit can be easily detected, as all changes output a different digest. A digest of a dataset serves as a means

of unique file identification; unique persistent identifiers (PID) can be used as pointers to a data location. Any change to the PID is visible and trackable [83].

Cryptographic hash functions are heavily used in the blockchain. They are used to create unique transactions and block IDs, to provide proofs of inclusion – meaning to prove if a transaction is contained in a block – and, most importantly, to achieve decentralized consensus among the participants. In particular, Bitcoin makes use of the SHA256 and RIPEMD160 hash functions. That said, cryptographic hash functions play an important role in the blockchain ecosystem.

2.3 Symmetric-key cryptography

In a symmetric cryptosystem two parties share a common *secret key* agreed prior to communication. The key is used for both encryption and decryption. When a party wants to securely send a message she has to use the secret key to encrypt it and then sent it. The receiver uses the same secret key to decrypt and recover the message. More formally, a symmetric cryptosystem is composed of the following algorithms [101, 102]:

- A **key generation algorithm** $\mathcal{G}$ that takes as input a security parameter 1^n and outputs a key k.
- An **encryption algorithm** $\mathcal{E}$ that takes as input a key k and a plaintext m and outputs a ciphertext c.
- A **decryption algorithm** $\mathcal{D}$ that takes as input a key k and a ciphertext c and outputs a plaintext m.

The set of all possible keys which can derive from the key generation algorithm $\mathcal{G}$ is called the *key space* $\mathcal{K}$. Respectively, the set of all possible plaintext is called the *plaintext message space* denoted $\mathcal{M}$, and the set of all possible ciphertexts is called *ciphertext message space* denoted $\mathcal{C}$. In practice, keys are usually of some fixed length; 256-bit keys are very common. On the other hand, messages and ciphertexts can be of arbitrary length. For example, a message may be a video or a music file or even a single bit.

A symmetric cryptosystem must satisfy the *correctness* property: for all $m \in \mathcal{M}$ and $k \in \mathcal{K}$, it holds that:

$$\mathcal{D}_k(\mathcal{E}_k(m)) = m$$

Any deterministic cryptosystem can not be secure [101,102]. For this reason, randomness is essential to any encryption scheme.

In the following chapters the notions of cipher and symmetric encryption scheme or symmetric scheme are identical and will be used interchangeably.

2.3.1 Block Ciphers

A block cipher is a deterministic algorithm that encrypts blocks – a sequence of bits – of fixed length. The plaintext and the ciphertext is always of the same size. The size is fixed

by the block cipher and is called the *block size*. The message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ are the same finite set $\{0,1\}^n$ and the key space is $\mathcal{K} = \{0,1\}^l$.

Formally, a block cipher is a keyed permutation function $F : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^n$ where n is the block size and l the key size. Usually, the block size and the key size are the same; common key and block sizes range from 128 to 512 bits. The function F must be *one-to-one* for every key $k \in \mathcal{K}$ and both F and its inverse F^{-1} should be easily computed; there is polynomial-time algorithm that given k computes F and F^{-1}. The number of total permutation a block cipher can produce is $2^n!$.

The security of a block cipher is much stronger than semantic security [30]: A permutation produced by F is indistinguishable from a random permutation. Cryptographic schemes based on block ciphers with rigorous proofs of security can, therefore, be constructed.

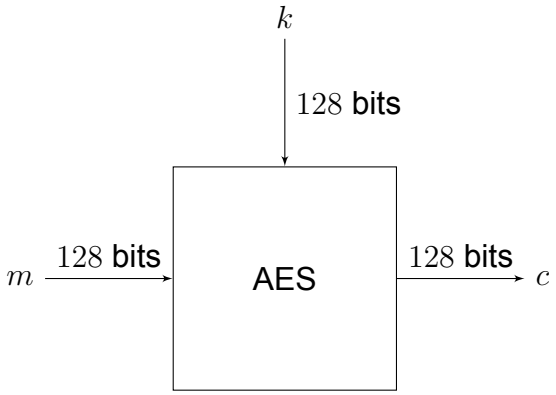


Figure 1: Block cipher AES

2.3.1.1 Modes of Operation [101]

A mode of operation is a way to encrypt arbitrary-length messages using block ciphers of fixed message length. A message is broken into l blocks of size n, the block cipher operating on each block. If the size of the last block is less than the block size, the block is padded to a full size block. In most modes of operation, a random initial vector (IV) of length n is required. The IV ensures distinct ciphertexts for each encryption even for the same plaintext and key. The IV is not secret as it is crucial for completing decryption. For a mode to be secure, the IV must be distinct and random in each encryption [101]

Several modes of operation are presented below. These modes are vulnerable either to padding oracle attacks [168] – attacks that exploit the padding validation to decrypt the ciphertext – or malleable [58] – the property of transforming one ciphertext into another which decrypts to a different plaintext. To countermeasure padding oracle attacks, or changes to the ciphertext, message authentication [101] should be used along with encryption to detect tampering with the ciphertext. Notable modes that combine encryption with authentication are Counter with CBC-MAC (CCM), Offset Codebook Mode (OCB) and Galois/Counter Mode (GCM).

Electronic Codebook (ECB)

The Electronic Codebook (ECB) is the simplest mode of operation. Given a plaintext $m = m_1, m_2, \ldots, m_l$ each block is encrypted separately. The ciphertext is $c = F_k(m_1), F_k(m_2), \ldots, F_k(m_l)$.

ECB mode is not secure and should never be used.

Cipher Block Chaining (CBC)

The Cipher Block Chaining (CBC) mode outputs a sequence of cipher blocks where each plaintext block is XORed with the previous ciphertext block. Each ciphertext is dependent on all preceding blocks. First, a random IV of length n is selected. The first plaintext block is XORed with the IV. The second block is XORed with the first cipher block c_1. In general, let $c_0 = IV$ then $c_i = F_k(c_{i-1} \oplus m_i)$, $\forall i \in [1, l]$.

The main drawback of CBC mode is that encryption and decryption are sequential – meaning that they cannot be parallelized.

Output Feedback (OFB)

The Output Feedback (OFB) mode uses the block cipher to generate a pseudorandom stream which is then XORed with the plaintext. The stream is generated only by repeatedly encrypting the IV independently of the plaintext. The stream is generated as follows. Set the i-th block of the stream to $r_i = F_k(r_{i-1})$ where $r_0 = IV \xleftarrow{r} \{0,1\}^n$. The ciphertext is produced by XORing the plaintext with the appropriate stream block; that is, $c_i = m_i \oplus r_i$.

As in CBC, the encryption and decryption cannot be parallelized. On the other hand, the pseudorandom stream can be computed independently of the actual message encryption and can be prepared ahead of time.

Counter (CTR)

The Counter (CTR) mode, like OFB, also generates a pseudorandom stream. First, a random $CTR \xleftarrow{r} \{0,1\}^n$ is chosen – much like the IV in previous modes. The stream is generated as $r_i = F_k(CTR + i)$ where $r_0 = CTR$. Same as before, the ciphertext is computed as $c_i = m_i \oplus r_i$.

CTR mode has many advantages. One of the main advantages of CTR mode is that both encryption and decryption can be fully parallelized. Secondly, as with OFB, the stream can be computed in advance. Finally, it is possible to decrypt the i-th block of the ciphertext without having to decrypt any other cipher block. This property is called random access.

2.4 Public Key Cryptography

As we see in 2.3 a secret key needs to be agreed upon prior to communication. In 1976, Whitfield Diffie and Martin Hellman published a paper called New Directions in Cryptography [56] which changed completely the way we communicate. The two cryptographers proposed a protocol that enabled two parties, having no prior communication, to establish a secret key over an insecure channel in the presence of eavesdropping adversaries. The protocol uses two keys, one for encryption and one for decryption. The encryption key is called the *public key* and the decryption key is called the *secret key* (or private). Every

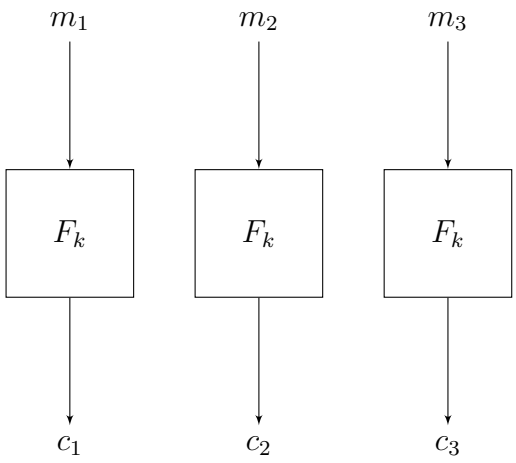


Figure 2: ECB mode

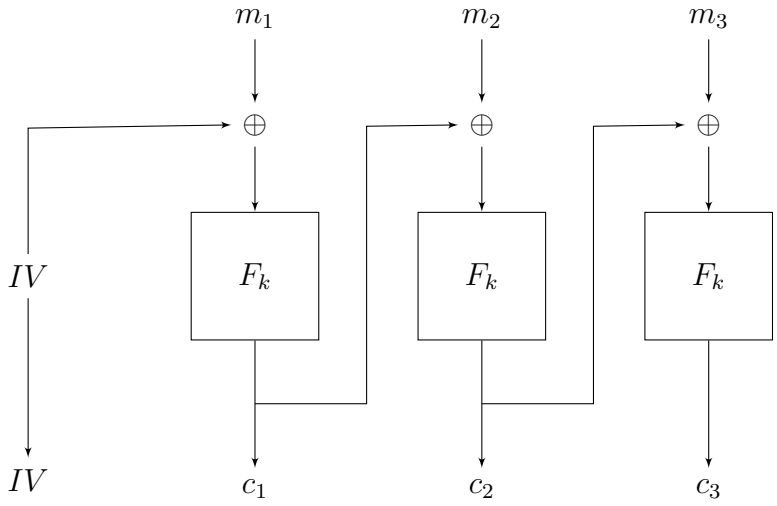


Figure 3: CBC mode

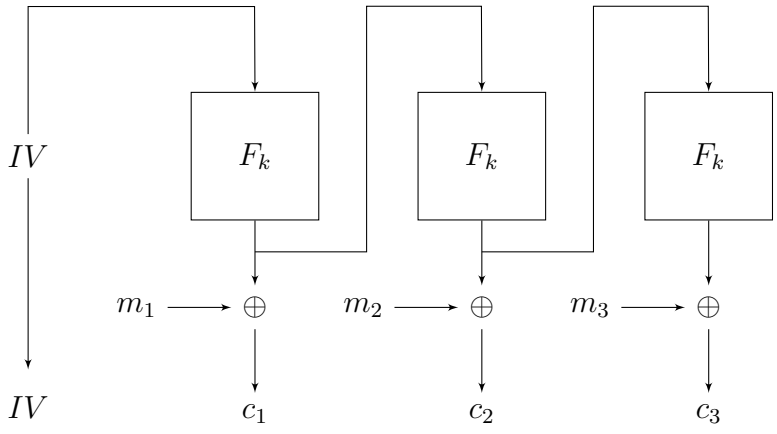


Figure 4: OFB mode

party has a key pair, consisting of a public and a secret key. The public is available to anyone who wants to sent an encrypted message to the key holder – the receiver may post the public key online beforehand. The receiver of the message decrypts the ciphertext with the use of her private key. Only the owner of the private key can decrypt a message that was encrypted using the corresponding public key. Any key pair is essentially an identity and the blockchain smartly utilizes it to provide anonymous identities to the users

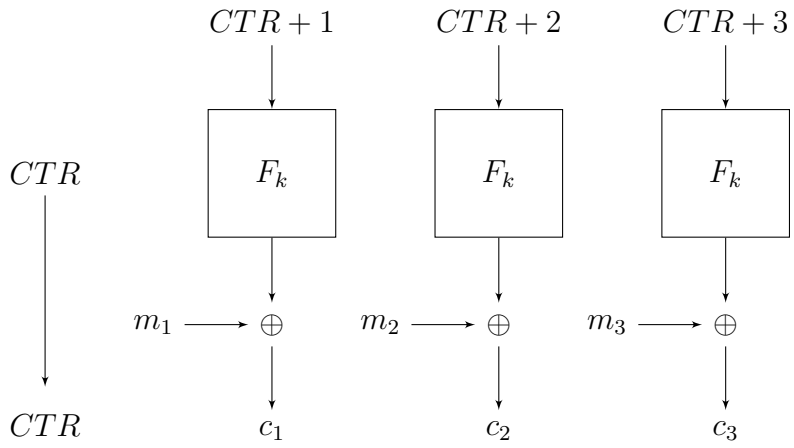


Figure 5: CTR mode

of the system.

A public-key encryption scheme is composed of the following probabilistic, polynomial-time algorithms [101, 102]:

- The **key generation algorithm** $\mathcal{G}$: Takes as input a security parameter 1^n it and outputs a key pair (p_k, s_k).
- The **encryption algorithm** $\mathcal{E}$: Takes as input a public key p_k and a plaintext m and outputs a ciphertext c.
- The **decryption algorithm** $\mathcal{D}$: Takes as input a private key s_k and a ciphertext c and outputs a plaintext m.

Likewise, a public-key cryptosystem must satisfy the correctness property: for all $m \in \mathcal{M}$ and $(p_k, s_k) \in \mathcal{K}$, it holds that:

$$\mathcal{D}_{s_k}(\mathcal{E}_{p_k}(m)) = m$$

Many important and widely used public-key schemes base their security on mathematical problems that, under certain conditions, are assumed to be hard; they cannot be solved in polynomial time. Such problems are the discrete logarithm and the factoring problem.

2.4.1 Discrete Logarithm and Diffie-Hellman Assumptions [101, 102]

In cryptography a system is considered to be secure if it is provable secure. There are cryptosystems which are perfectly secure [157], like the *one-time pad* [157], and others that perfect security cannot be achieved. In the second case security can be proven by computational security assumptions, in other words a reduction to a particular problem which under certain conditions is assumed to be hard to solve in polynomial-time. Significant cryptographic schemes like the Diffie-Hellman key exchange and El Gamal encryption and signature protocol are built according to the discrete log computational hardness assumption.

Definition 2.4.1.1. Let $\mathbb{G}$ be a cyclic group of order q and let g be a generator of $\mathbb{G}$. The *discrete log problem* (DLOG) is as follows: given a random element $h \in \mathbb{G}$, find an integer $x \in \mathbb{Z}_q$ such that $g^x = h$.

Definition 2.4.1.2. The *computational Diffie-Hellman problem* (CDH) is as follows: given a cyclic group $\mathbb{G}$ of order q, a generator $g \in \mathbb{G}$, g^a and g^b where $a, b \xleftarrow{r} \mathbb{Z}_q$, compute g^{ab}.

Definition 2.4.1.3. The *decisional Diffie-Hellman problem* (DDH) is as follows: given a cyclic group $\mathbb{G}$ of order q, a generator $g \in \mathbb{G}$, and g^a, g^b, g^c where $a, b, c \xleftarrow{r} \mathbb{Z}_q$, decide if $c = ab$ or $c \xleftarrow{r} \mathbb{Z}_q$.

The DLOG problem is believed to be hard for specific group families of $\mathbb{G}$. For example, solving the DLOG for the subgroup of order q of $\mathbb{Z}_p^*$, where p and q are primes of size at least 2048-bits and 256-bits respectively, is considered infeasible. This assumption is called the *discrete log assumption*.

The DDH problem is weaker than the CDH problem. It holds that $DDH \leq CDH$; if an adversary can solve CDH she can compute g^{ab} and compare it to g^c and easily solve the DDH.

2.5 Digital signatures

A *digital signature* is a fundamental cryptographic primitive. It is considered to be the equivalent to a handwritten signature. It is a scheme that demonstrates the authenticity of digital messages or documents.

In a digital signature scheme, each party holds a unique key pair (p_k, s_k). A message m is uniquely signed with the *signing key* s_k and the *verification key* p_k is used to verify the signature. Only someone who knows s_k can sign a message, but all parties that have access to p_k can verify a signature.

Digital signatures have the following important properties:

- **Authentication**: The message is signed by a known sender
- **Non-repudiation**: The sender cannot deny sending the message
- **Integrity**: The message is not altered in transit

Digital signatures are commonly used for software distribution and financial transactions and in cases where forgery detection is important. The blockchain uses digital signatures to provide asset ownership; the rightful owner signs the transaction to prove her possession of the asset.

A digital signature scheme is composed of the following probabilistic polynomial-time algorithms [101, 102]:

- The key **generation algorithm** Gen: Takes as input a security parameter 1^n and outputs a key pair (p_k, s_k).
- A **signing algorithm** $Sign$: Takes a signing key s_k and a message m and produce a digital signature σ of m
- A **deterministic verification** algorithm $Verify$: Takes a verification key p_k and a signature σ. It outputs $b = 1$ or $b = 0$ ($true$ or $false$) to indicate if the signature is valid.

The primary goal of digital signatures is *unforgeability*; an adversary cannot create a new valid message-signature pair without the corresponding signing key.

2.5.1 El Gamal Signature Scheme

The *El Gamal Signature Scheme* [65] is a digital signature scheme based on the difficulty of the discrete logarithm problem (§ 2.4.1).

The scheme works as follows:

- **Key Generation**:
  1. Choose a cryptographic hash function H where $H : \{0,1\}^* \to \{0,1\}^n$
  2. Choose a prime p such that the DLOG problem is difficult
  3. Find a generator g of the group $\mathbb{Z}_p^*$
  4. Choose randomly $x \xleftarrow{r} \mathbb{Z}_q$
  5. Compute $h = g^x mod p$
  6. Return (h, x) where h is the public key and x the private key
- **Sign**: Sign a message $m \in \{0,1\}^*$ with a private key x
  1. Choose randomly $k \xleftarrow{r} \mathbb{Z}_q$ such that $gcd(k, p-1) = 1$
  2. Compute $r = g^k mod p$
  3. Compute $s = k^{-1}(H(m) + xr) mod (p-1)$
  4. Return the signature (r, s)
- **Verify**: Verify a signature (r, s) of the message m with a public key h
  1. Verify that $r \in \mathbb{Z}_q$ and $s \in \mathbb{Z}_q$. Else ouput 0.
  2. Compute $v = H(m)$
  3. If $v \stackrel{?}{=} (h^r r^s) mod p$ then output 1 else 0.

2.5.2 Digital Signature Algorithm (DSA)

The *Digital Signature Algorithm* (DSA) [78] is a variant of the El Gamal signature scheme. The DSA was standardized by the National Institute of Standards and Technology (NIST). The main advantage of DSA over El Gamal Signature is the size of the signature. DSA produces signatures of 320-bit in size, in comparison to El Gamal where at least 2048-bit signatures are required for it to be secure according to contemporary security standards [34].

The algorithm works as follows [34]:

- **Key Generation**:
  1. Choose a cryptographic hash function H where $H : \{0,1\}^* \to \{0,1\}^n$
  2. Choose a prime q of size n
  3. Choose a prime p such that $p-1$ is a multiple of q of size l. The size of l must be a multiple of 64 between 512 and $1,024$
  4. Choose randomly $a \xleftarrow{r} \mathbb{Z}_p$

5. Compute $g = a^{(p-1)/q} mod p$
6. Choose randomly $x \xleftarrow{r} \mathbb{Z}_q$
7. Compute $h = g^x mod p$
8. Return $((p, q, g), h, x)$ where (p, q, g) is the public parameters of the algorithm, h is the public key and x the private key.

- **Sign**: Sign a message $m \in \{0, 1\}^*$ with a private key x
  1. Choose randomly $k \xleftarrow{r} \mathbb{Z}_q$
  2. Compute $r = (g^k mod p) mod q$
  3. Compute $s = k^{-1}(H(m) + xr) mod q$
  4. Return the signature (r, s)
- **Verify**: Verify a signature (r, s) of the message m with a public key h
  1. Verify that $r \in \mathbb{Z}_q$ and $s \in \mathbb{Z}_q$. Else ouput 0.
  2. Calculate $w = s^{-1} mod q$
  3. Calculate $u_1 = (H(m)w) mod q$
  4. Calculate $u_2 = (rw) mod q$
  5. Calculate $v = ((g^{u_1} h^{u_2}) mod p) mod q$
  6. If $v \stackrel{?}{=} r$ then output 1 else 0.

2.6 Elliptic-curves

Elliptic curves (ECC) play an import role in cryptography. All cryptosystems presented in previous chapters are built over multiplicative groups of integers modulo a sufficient large prime p [30, 102]. Elliptic curves are additive *abelian groups* [102]; a group satisfying associativity, commutativity, and existence of identity element and of inverse element under the addition group operation [88].

There are two apparent benefits for using curves over modular groups [102]. The first is cost efficiency. In practice, cryptosystems that their security depends on the discrete log problem or the factorisation problem on a modular group, such as the El Gamal or the RSA, the primes have to be at least 2048 bits for the system to be secure. On the other hand, elliptic curves offer similar security using much smaller key sizes. A 233-bit elliptic curve key gives the same level of security as a 2,240-bit RSA key [114, 165] and a 333-bit elliptic curve key as a 4096-bit RSA key [27]. The second reason is that a way to generalize the attacks against the discrete logarithm problem on a modular group to elliptic curves [102] has not been found yet.

Cryptography is primarily interested in elliptic curves over $\mathbb{F}_p$: the field of integers modulo p, where p is a prime number. An elliptic curve E over $\mathbb{F}_p$ is defined by the equation of the form [88, 170]:

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{F}_p$. A pair (x, y) where $x, y \in \mathbb{F}_p$ is a point on the curve if it satisfies the equation where E is defined. There is a special point, called *point at infinity* and denoted by ∞, where is also on the curve and serve as the *identity element* [88].

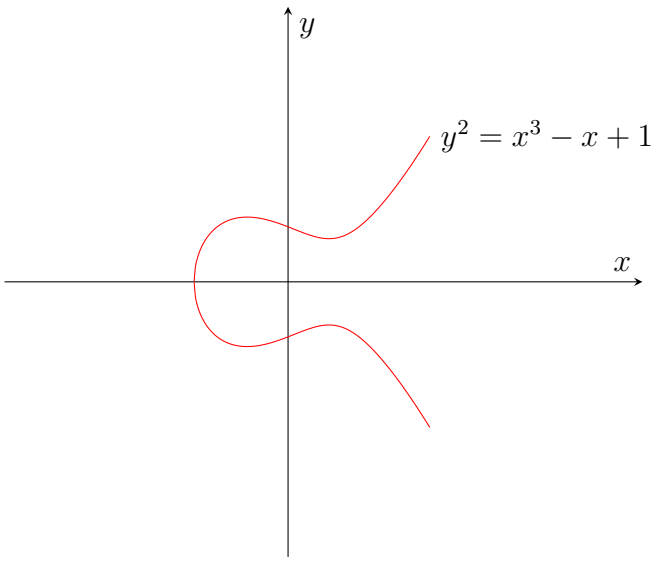


Figure 6: Elliptic curve

2.6.1 Points addition [170]

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points on the elliptic curve E where $P \neq Q$ and $P, Q \neq \infty$. Any two points on a curve add to produce a third point on the curve. A third point $R = (x_3, y_3)$ on the curve E is defined as:

$$R = P + Q$$

We can find the third point R as follows:

1. Draw the line L through P and Q.
2. Find the third point $-R$ that L intersects with.
3. Reflect $-R$ across the x-axis to obtain R

The slope of the line L can be found as

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

and assuming $x_1 \neq x_2$ the equation of L is

$$y = m(x - x_1) + y_1$$

To find the intersection with E, substitute y to get

$$(m(x - x_1) + y_1)^2 = x^3 + ax + b$$

The three roots of cubic polynomial correspond to the three points of intersection of L with E. As we already know the two of the tree roots, since P and Q are points on both L and E, finding the third root is easy. For a cubic polynomial $x^3 + ax^2 + bx + c$ with roots r, s, t it holds that [170]

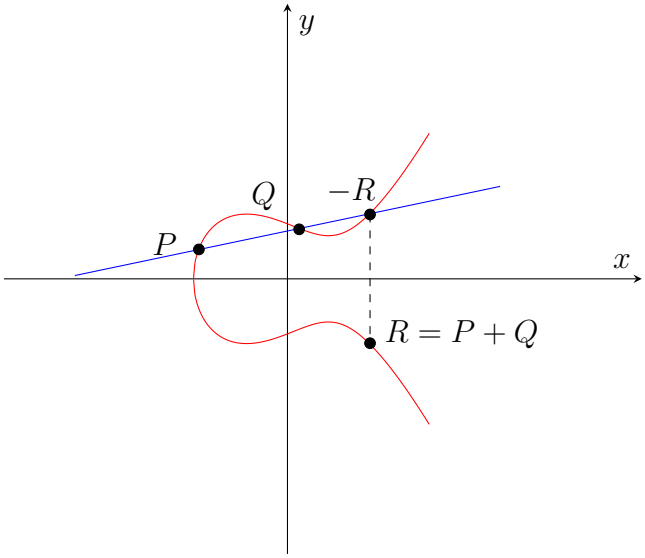


Figure 7: Addition of points on elliptic curves

$$x^3 + ax^2 + bx + c = (x - r)(x - s)(x - t) = x^3 - (r + s + t)x^2 + \dots$$

Therefore,

$$r + s + t = -a$$

Knowing the two roots r, s we can recover the third as $t = -a - r - s$.

So, to find the third point where L intersects with E we obtain

$$x = m^2 + -x_1 - x_2$$

and

$$y = m(x - x_1) + y_1$$

Finally, to reflect $-R$ to the x-axis to obtain R we change the sign of y_1. The final equations that give the third point are

$$x_3 = m^2 + -x_1 - x_2, \qquad y_3 = m(x_1 - x_3) - y_1$$

2.6.2 Points doubling [170]

Let $P = (x_1, y_1)$ a point on the elliptic curve E where $P \neq -P$ and $P \neq \infty$. Then $2P = (x_3, y_3)$. Unlike 2.6.1 where there are two points, in point doubling we have only one point, thus we cannot draw the line L. In that case, we use the tangent line L to the curve E at point P.

The slope of L can be found as:

$$m = \frac{dy}{dx} = \frac{3x_1^2 + a}{2y_1}$$

Assuming $y_1 \neq 0$ the equation of L is

$$y = m(x - x_1) + y_1$$

Therefore, proceeding as in 2.6.1, we obtain

$$x_3 = m^2 + -2x_1, \qquad y_3 = m(x_1 - x_3) - y_1$$

2.6.3 Elliptic curve discrete logarithm problem (ECDLP)

Cyclic subgroups of some elliptic curve groups can be used to implement secure systems based on the discrete logarithm problem. The DLOG is assumed to be hard, like in multiplicative groups.

Definition 2.6.3.1. The elliptic curve discrete logarithm problem (ECDLP) is as follows: given an elliptic curve E over a finite field $\mathbb{F}_p$ and two points $P, Q \in E$, find an integer $d \in \mathbb{Z}_p$ such that $Q = dP$.

The parameters that describe an elliptic curve E defined over a finite field $\mathbb{F}_p$, a base (generator) point $P \in E$, and its order q should be chosen carefully so that the ECDLP is resistant to all known attacks.

2.6.4 Key generation [88]

Let E be an elliptic curve over $\mathbb{F}_p$ and P a generator point in E of order q. The key pair generation algorithm is defined as follows:

1. Select a random integer $d \xleftarrow{r} \in \mathbb{Z}_q$
2. Compute $Q = dP$
3. Return the tuple (Q, d) where Q is the public key and d the secret key

The prime p, the curve E the generator point P, its order q and the public key Q are public. Finding the secret key d is equivalent to solving the ECDLP.

2.6.5 Elliptic Curve Digital Signature Algorithm (ECDSA)

The *Elliptic Curve Digital Signature Algorithm* (ECDSA) [98] is a variant of the Digital Signature Algorithm (DSA) which uses elliptic Curves. It is the most widely adopted elliptic curve-based signature scheme [88].

Let E be an elliptic curve over $\mathbb{F}_p$, a generator point P of order q and H a cryptographic hash function where $H : \{0,1\}^* \to \{0,1\}^q$. The ECDSA is defined as follows [88]:

- **Key generation**: Run elliptic curve key generation algorithm defined in 2.6.4 and get a key pair (Q, d).
- Sign: Sign a message m with a private key d
  1. Choose randomly $k \xleftarrow{r} \mathbb{Z}_q$
  2. Compute the point $kP = (x_1, y_1)$
  3. Compute $r = x_1 mod q$. If $r \stackrel{?}{=} 0$ then go to step 1
  4. Compute $e = H(m)$
  5. Compute $s = k^{-1}(e + dr) mod q$. If $s \stackrel{?}{=} 0$ then go to step 1
  6. Return the signature (r, s)
- **Verify**: Verifies a signature (r, s) of the message m with a public key Q
  1. Verify that $r \in \mathbb{Z}_q$ and $s \in \mathbb{Z}_q$. Else ouput 0.
  2. Compute $e = H(m)$
  3. Compute $w = s^{-1} mod q$
  4. Compute $u_1 = (ew) mod q$ and $u_2 = (rw) mod q$
  5. Compute $X = u_1 P + u_2 Q$
  6. If $X \stackrel{?}{=} \infty$ output 0
  7. Take x_1 cordinate of X and compute $v = x_1 mod q$
  8. If $v \stackrel{?}{=} r$ then output 1 else 0.

Blockchain uses elliptic curves for key pair generation and transaction signing (ECDSA). Bitcoin and Ethereum use a specific elliptic curve for ECDSA [98] whose parameters are defined in the secp256k1 standard [142].

2.7 Homomorphic Encryption

Homomorphic encryption allows to perform specific types of computation on encrypted data. Computations performed over a ciphertext return encrypted results. When the results are decrypted they match the results of the operations as if they had been performed on the plaintext. An encryption scheme is called homomorphic when it has the homomorphic property. In particular, if for all $m_1, m_2 \in \mathcal{M}$ and $k \in \mathcal{K}$ (secret key or public key):

$$Enc_k(m_1) \otimes Enc_k(m_2) = Enc_k(m_1 \oplus m_2)$$

then the encryption scheme is homomorphic.

Homomorphic encryption schemes are by nature *malleable* and have weaker security properties than non-homomorphic schemes. Various known encryption schemes are homomorphic such as the unpadded RSA and El Gamal.

2.8 Zero Knowledge Proofs [102]

A *proof of knowledge* is a protocol that enables one party to convince another of the validity of a *statement*. In a *zero-knowledge proof* [82] this is accomplished without revealing any information beyond the legitimacy of the proof [102]. In other words, one party can prove to another party that a given statement is true, without conveying any information apart from the fact that the statement is indeed true.

Let $\mathcal{P}$ be a prover and $\mathcal{V}$ the verifier. $\mathcal{P}$ must convince $\mathcal{V}$ that she has some knowledge of a statement x without explicitly stating what she knows. We call this knowledge a *witness* w. Both parties are aware of a predicate R that will attest to w being a valid witness to x [102]. In general,

- The predicate R is assumed to be polynomial-time computable.
- The prover $\mathcal{P}$ has R, x, and w such that $R(x, w) = 1$. She wishes to prove possesion of w by producing a proof of knowledge π.
- The verifier $\mathcal{V}$ has R, x, and π.
- Given R it is hard to find a corresponding w such as $R(x, w) = 1$
- The prover $\mathcal{P}$ is unwilling to reveal w; otherwise the solution is trivial.
- The verifier $\mathcal{V}$ can efficiently check the validity of π.

2.8.1 Examples

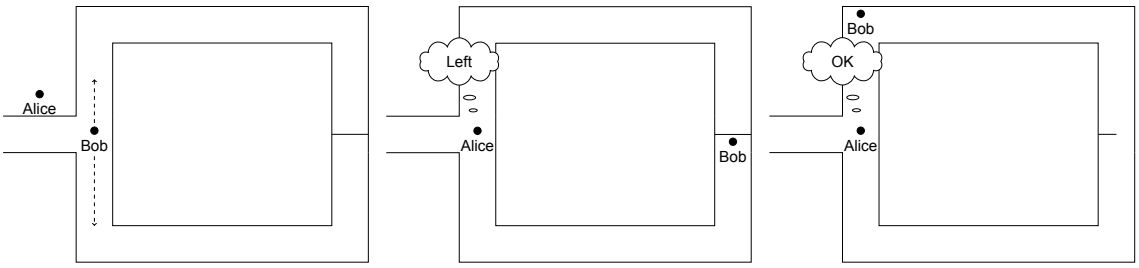
2.8.1.1 The strange cave of Ali Baba

Let's bring in all cryptographers' old friends, Alice and Bob, and let them play the following game: at the bottom of the cave [143] of figure 8 there is a magic door that can only open with a secret password. The cave has one entrance on one side and a magic door blocking the entrance on the opposite side. Bob knows the secret password and he wants to prove that to Alice without revealing the password. Bob proposes the following game to prove that he can open the magic door: Alice has to wait outside of the cave, while Bob enters the cave and takes either the left or the right path. Alice cannot see which path Bob is taking. Then, Alice enters; standing at the entrance of the cave, she calls Bob to come out from either the left or the right passage. Bob does so, using the secret password if necessary.

If Bob does not know the secret password he has $1/2$ change of guessing correctly as Alice chooses at random which path Bob is taking. If this process is repeated k times the probability of Bob convincing Alice that he knows the secret password without actually knowing is exponentially reduced – at most $1/2^k$.

2.8.1.2 The colour-blind friend

Now, suppose that Bob is color-blind [39] and Alice keeps two balls of the same size in her hands, one red and one green. To Bob they seem identical and he is not sure which one is which. Alice wants to prove him that indeed they are of different colour, without



(a) Alice stands outside and Bob choose randomly a path, left or right

(b) Alice calls to Bob, asking him to come out either the left or the right passage

(c) Bob complies appearing at the exit Alices names

Figure 8: Ali Baba Cave

revealing which one is red and which is green. To prove her knowledge, she tells Bob to hold the two balls, one on each hand. Alice can see at this point which ball is in which hand. Next, he can either switch hands behind his back or leave them be. Finally, he reveals them. Alice can say with certainty whether he switched hands or not. Have the balls had the same color, they would have been indistinguishable. There would have been no way Alice could guess the right answer with probability higher than $1/2$. If they repeat this k times the probability that Alice succeeds when the balls are identical is at most $1/2^k$.

2.8.2 Formal Definition [102]

Let $< \mathcal{P}, \mathcal{V} >$ be a pair of interactive programs. Define $\text{out}^{\mathcal{P}}_{\mathcal{P},\mathcal{V}}(x, w, z)$ to be the output of $\mathcal{P}$ when both $\mathcal{P}$ and $\mathcal{V}$ are executed with the public input x and private inputs w and z ($\mathcal{P}$ determines w and $\mathcal{V}$ choose z); $\text{out}^{\mathcal{V}}_{\mathcal{P},\mathcal{V}}(x, w, z)$ is similar defined for $\mathcal{V}$. The PPT interactive protocol $< \mathcal{P}, \mathcal{V} >$ is a *zero-knowledge proof* for a language $L \in NP$ with knowledge error k and zero-knowledge distance ε if the following properties hold [102].

1. **Completeness:** If $x \in L$ and $R(x, w) = 1$ for some witness w, then $\text{out}^{\mathcal{V}}_{\mathcal{P},\mathcal{V}}(x, w, z)$ = 1 for all string z with overwhelming probability in v.

2. **Soundness:** For any polynomial-time program $\mathcal{P}^*$ define for arbitrary x, w, z,

$$\pi_{x,w,z} = \text{Prob}[\text{out}^{\mathcal{V}}_{\mathcal{P}^*,\mathcal{V}}(x, w, z) = 1].$$

   A protocol $< \mathcal{P}, \mathcal{V} >$ satisfies soundness if there are non-negligible functions $s(v), q(v)$ such for all $\mathcal{P}^*$ here exists a probabilistic Turing machine (PTM) program K, called a knowledge extractor with the following property. Suppose that

$$\widetilde{\pi} = \text{Prob}[K(x, w, z) = w^{'} : R(x, x^{'}) = 1].$$

   Then it holds that $\pi_{x,w,z} \geq s(|x|)$ implies that $\widetilde{\pi}_{x,w,z} \geq q(|x|)$.

3. **(Statistical) Zero-knowledge:** For each polynomial-time program $\mathcal{V}^*$, there is a PTM program S, called the *simulator*, such that for all x, w with $R(x, w) = 1$, the random variables $S(x, z)$ and $\text{out}^{\mathcal{V}^*}_{\mathcal{P},\mathcal{V}^*}(x, w, z)$ are statistically indistinguishable for all strings z:

$$\forall \mathcal{A} \left| \text{Prob}[\mathcal{A}(S(x, z) = 1)] - \text{Prob}[\mathcal{A}(\text{out}^{\mathcal{V}^*}_{\mathcal{P},\mathcal{V}^*}(x, w, z)) = 1] \right| < \varepsilon.$$

Completeness is very similar to correctness. Assuming both the prover and verifier follow the protocol faithfully, completeness guarantees that the protocol will succeed with a sufficiently high probability. Soundness ensures that if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability. Intuitively, statistical zero-knowledge is a property that prohibits a verifier from extracting information from an honest prover. A weaker version of zero-knowledge is *honest-verifier zero-knowledge* (HVZK) where it is assumed that the verifier executes the protocol faithfully, but makes additional computations.

2.8.3 Verifiable Computation (VC)

In devices such as mobiles or IoT devices where the computational power is often limited, the need to outsource computation to one or more powerful workers on the cloud emerges. Yet, confidence is required for the working entity to assure computation is done properly. The client should be able to verify the correctness of the output. This way, the client is protected from malicious or malfunctioning workers, and the legitimate worker is no longer accountable for the computation [138]. This scheme is called *public verifiable computation* (VC) [138].

In a public verifiable computation scheme the client outsources the evaluation of a function F on input u – for example, a query. The client then can verify the correctness of the computation of $F(u)$ doing less work than the initial computation of $F(u)$. The outsource function F can take two inputs u and w, where w is the worker's private input – for example, a data set. In this case, the scheme is a *Zero-Knowledge Verifiable Computation* (or non-interactive zero knowledge (NIZK) proof [29]) if the client learns nothing about the worker's input except the computation's output [138].

Such schemes are not interactive, which means that no interaction is necessary between prover and verifier (worker and client), and a *common reference string* (CRS) shared between them is enough to achieve computational zero-knowledge [29].

Formally, as defined by Parno et. al [138], a public Zero-Knowledge verifiable computation scheme $\mathcal{VC}$ consists of a set of three polynomial-time algorithms:

- $(EK_F, VK_F) \leftarrow$ KeyGen$(F, 1^\lambda)$: The **randomized key generation algorithm**. It takes as input the outsourced function F and a security parameter λ; it outputs a public evaluation key EK_F and a public verification key VK_F.
- $(y, \pi_y) \leftarrow$ Compute(EK_F, u): The **deterministic worker algorithm**. It takes as input the public evaluation key EK_F and a public input u; it outputs $y \leftarrow F(u, w)$, where w is an auxiliary private input, and a proof π of y's correctness.
- $\{0, 1\} \leftarrow$ Verify(VK_F, u, y, π_y): The **deterministic verification algorithm**. It takes as input the public verification key VK_F, the public input u, the output y and the proof π_y; It outputs 1 if $F(u) = y$ and 0 otherwise.

2.8.4 zk-SNARKs

A *Non-interactive Zero-Knowledge proof* (NIKZ) [29] is a zero-knowledge proof where sharing a common reference string (CRS) between the prover and the verifier ahead of time is enough to implement zero knowledge protocols without the need for interaction

between the participants of the protocol. The *zkSNARKs* protocols are non-interactive zero-knowledge proof protocols which have some extra properties regarding the size of proof and the verification time.

The acronym zkSNARK stands for *'Zero-Knowledge Succinct Non-Interactive Argument of Knowledge'* in which each individual part – informally defined – have the following meaning [21, 25, 120, 147, 152]:

- **Zero-Knowledge**: The verifier learns nothing but the validity of the computation
- **Succinct**: The proofs are sort and easy to verify in comparison to the actual computation (constant size, and polynomial verifiable in the size of the proof).
- **Non-Interactive**: There is no or little interaction in one step. Anyone can verify the proof, no need for new interaction (publicly verifiable).
- **ARgument**: Soundness (§ 2.8.2) holds only against computationally bounded provers.
- **of Knowledge**: If the verifier accepts a proof output by a computationally bounded prover, then the prover has a witness for the given instance.

The first zkSNARKs constructions where inspired by the *PCP* theorem allowing faster and shorter proofs [120]. In this section an analysis of the zk-SNARK model proposed by Gennaro et al [80] and the Pinnochio protocol proposed by Parno et al [138] is attempted, without getting into rigorous mathematical proofs, security assumptions or implementations. The purpose of this chapter is to provide a high level understanding of zkSNARKs and their core mechanisms.

2.8.5 Main idea

In order to be able to construct a zkSNARKs for a computation problem it has to be encoded in a specific form, an equivalent to the original problem, from which zkSNARKs can derive. This form is called *Quadratic Arithmetic Program* (QAP).

At a high level, zkSNARKs consists of four basic steps (Figure 9):

1. Computation to Arithmetic circuit [138]
2. Arithmetic circuit to Rank 1 Constraint System (R1CS) [80]
3. R1CS to Quadratic Span Program (QAP) [80]
4. QAP to zkSNARK [138]

Figure 9: Steps of zk-SNARK

2.8.5.1 Arithmetic circuits

An *arithmetic circuit* C over a finite field $\mathbb{F}_p$ is a circuit that contains only addition and multiplication gates. It takes as inputs elements in $\mathbb{F}_p$ and its gates output elements in $\mathbb{F}_p$ [21, 152]. The outputs are determined by the inputs which pass through the gates where their values are changed accordingly. Any program can be reduced to an arithmetic circuit [20, 137] and normally the circuit is associated to the function it computes – the outsource function upon which the prover works.

A legal or a *valid assignment* for an arithmetic circuit C is a tuple $(c_1, c_2, \ldots, c_n) \in \mathbb{F}_p$ such that $C(c_1, c_2, \ldots, c_k) = (c_{k+1}, c_2, \ldots, c_n)$ where k is the number of all input and $n - k$ the number of all outputs of the circuit. Given a circuit evaluation, the task of the prover is to convince the verifier that evaluations of intermediate gates exist, so that the circuit indeed produces such an output with such an input [137]. For example, if Alice wants to prove to Bob that she knows $c_1, c_2, c_3 \in \mathbb{F}_p$, such that $(c_1 c_2)(c_1 + c_3) = 7$, first she has to encode the computation to an arithmetic circuit C (the presentation of C is shown in Figure 10). Then she wants to prove that she knows a valid assignment $(c_1, c_2, c_3, c_4, c_5)$, where $c_4 = c_1 c_2$ and $c_5 = c_4(c_1 + c_3)$, such that $c_5 = 7$.

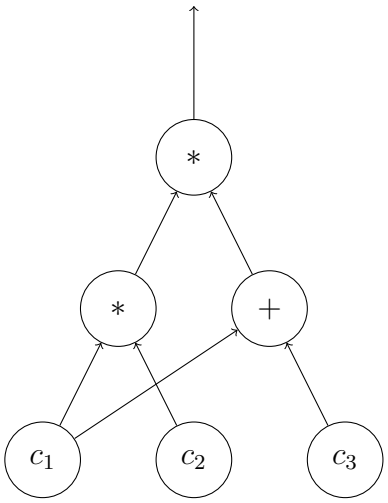


Figure 10: A simple arithmetic circuit

2.8.5.2 Quadratic Arithmetic Program (QAP) [120, 177]

Geranno et. al [80] showed how to efficiently encode computations to quadratic programs called Quadratic Arithmetic Programs (QAP), to obtain zk-SNARKs. QAPs play an important role as they enable the prover to construct the proof π where she claims that she knows a valid assignment of a circuit C.

A QAP $Q(C) = (L, R, O, T)$ for a given circuit C is a set of three polynomials

$$L = \sum_{i=1}^{m} c_i L_i \qquad R = \sum_{i=1}^{m} c_i R_i \qquad O = \sum_{i=1}^{m} c_i O_i \qquad (m \geq n)$$

and a target polynomial T such that T divides the polynomial:

$$P = LR - O$$

if and only if $(c_1, c_2, \ldots, c_n)$ is a valid assignment for C. The prover constructs the polynomial for its proof π and the verifier checks the divisibility of P by T. Identically, $P = TH$ for some polynomial H.

To translate the C into a QAP the wires and gates must be labeled in a specific way:

- Each multiplication gate has exactly two input wires: a left and a right wire
- Each multiplication gate has a unique label
- Addition gates are not labeled
- Outgoing wires to more than one gate are labeled as one wire
- Outgoing wires from an addition gate to a multiplication gate are not labeled; the inputs of an addition gate go directly to the multiplication gate

A label assignment of the circuit of Figure 10 is shown in Figure 11.

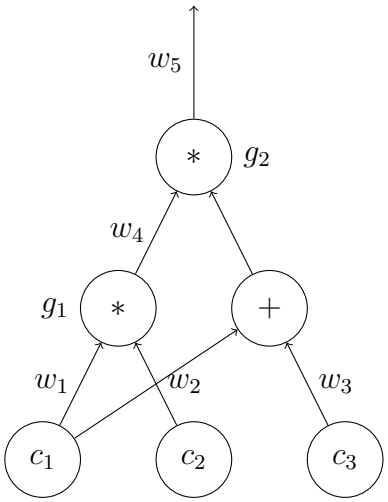


Figure 11: Circuit label assignment

The circuit is ready to be transformed into a QAP. Let M be the set that contains the indexes of the multiplication gates and W the set that contains the input and output wires. The points in M are called target points. For each gate $g \in M$ we construct a set of left, right, and output polynomials as follows: for each wire a polynomial is constructed in such way that it evaluates at zero on each target point except those that correspond to the multiplication gate.

QAP construction of circuit of Figure 11: Gate g_1, with target point 1, has w_1 as left wire, w_2 as right wire and w_4 as output label. As the polynomial on point 1 that corresponds to g_1 evaluates to 1 and on point 2 corresponding to g_2 evaluates to 0, we construct $L_1 = R_2 = O_4 = 2 - x$. Similar, for gate g_2 $L_4 = R_1 = R_3 = O_5 = x - 1$. Note that the wires w_2 and w_3 are both right inputs of g_2.

The wire polynomials:

$$L_1 = (2 - x) \qquad L_2 = 0 \qquad L_3 = 0 \qquad L_4 = (x - 1) \qquad L_5 = 0$$

$$R_1 = (x - 1) \qquad R_2 = (2 - x) \qquad R_3 = (x - 1) \qquad R_4 = 0 \qquad R_5 = 0$$

$$O_1 = 0 \qquad O_2 = 0 \qquad O_3 = 0 \qquad O_4 = (2 - x) \qquad O_5 = (x - 1)$$

The total left, right and output polynomials:

$$\begin{aligned} L &= \sum_{i=1}^{5} c_i L_i \\ &= c_1 L_1 + c_4 L_4 \\ &= c_1(2 - x) + c_4(x - 1) \end{aligned} \qquad \begin{aligned} R &= \sum_{i=1}^{5} c_i R_i \\ &= c_1 R_1 + c_2 R_2 + c_3 R_3 \\ &= c_1(x - 1) + c_2(2 - x) + c_3(x - 1) \end{aligned} \qquad \begin{aligned} O &= \sum_{i=1}^{5} c_i O_i \\ &= c_4 O_4 + c_5 O_5 \\ &= c_4(2 - x) + c_5(x - 1) \end{aligned}$$

If we evaluate these polynomials at target points 1 and 2 we get:

$$\begin{aligned} P(1) &= L(1)R(1) - O(1) \\ &= c_1 c_2 - c_4 \end{aligned} \qquad \begin{aligned} P(2) &= L(2)R(2) - O(2) \\ &= c_4(c_1 + c_3) - c_5 \end{aligned}$$

Hence, $T(x) = (x - 1)(x - 2)$ divides $P(x)$ if and only if $1, 2$ are roots of $P(x)$. Identically, the tuple $(c_1, c_2, \ldots, c_5)$ is a valid assignment for C.

3. BLOCKCHAIN

A *blockchain* is a *distributed transaction ledger* [128]. It consists of a continuously growing list of *blocks* which are linked, ordered, and immutable. Each block typically contains a *hash pointer* being a link to a previous block, a timestamp and a list of *transactions*. Transactions describe the transfer of assets from one entity to another. A distributed network is formed by the users of the system without central control. Each node of the network contains a local copy of the blockchain; the transaction ledger. To come to an agreement on the correct ledger, the network uses a decentralized *consensus* mechanism with which integrity is achieved and malicious activity is prevented.

Blockchains are potentially suitable for financial activities, the recording of events, medical records [13, 176], and other record management activities, such as identity management, transaction processing or documenting provenance. Blockchain can be seen as a distributed immutable, tamper-proof, and audit log that records all data transactions. As a result any attempt to tamper blockchain is immediately evident and easily detectable.

Bitcoin [128] is the first decentralized *cryptocurrency*, a monetary system without central authority invented by an unknown person or a group of people under the name of *Satoshi Nakamoto* [128]. Its ability to provide a solution to the *double spending problem* [100, 128] made it a powerful digital currency among physical currencies in less that 10 years, triggering an explosion of more than 800 other digital coins.

The goal of this chapter is to present of the basic operating principles of the blockchain, which are based on the foundation of cryptography – without getting to unnecessary details or rigorous mathematical proofs. As Bitcoin was the first decentralized blockchain system that put the bases for others to be made and involve [31, 36, 79], we mainly focus on blockchain internals. Notable variations of blockchains that support features of our interest are mentioned selectively further in this chapter.

3.1 History

Credit card transactions are the dominant payment method that is used on the web today [130]. This system is handled by a financial system involving processors, banks, credit card companies and other intermediaries. Normally a credit card transaction is realized as follows: first, the buyer sends over his credit card details to the merchant; then the merchant sends and validates the data in the financial system.

Buyers may not want to handle their credit card details to an unknown vendor over an insecure channel. Intermediate services – such as Paypal – sits between the buyer and the seller and the intermediate service approves the transaction and notify the seller. This allows the buyer to keep his anonymity and avoid security risks.

The idea of *digital cash* was first introduced by David Chaum in his paper [40], entitled “Blind Signatures for Untraceable Payments”, in 1983. In 1990 Chaum proposed the first off-line e-cash system [42] and founded DigiCash [41], an electronic money corporation that sent the first electronic payment in 1994.

Digital cash schemes are vulnerable to that is called double-spending [31, 79]. It comes up when the same digital token is spent more than once. Chaum found a way to both keep the system anonymous and prevent double-spendings with the use of *blind signatures* [40, 42]. Yet still, Chaum’s solution needed a centralized trusted oracle which validated the

transactions. Many cryptographers attempted to improve Chaum's scheme over the years. Okamoto and Ohta [171], for example, in 1996 implemented coin subdivisions with the use of *Merkle trees* [125].

About the same time, a group of cryptographers called *Cypherpunks* [94, 129] was formed and advocate the widespread use of strong cryptography and privacy-enhancing technologies [94] as a way to social and political change. Cypherpunks electronic mailing list, through which they communicating originally, was the predecessor to the mailing list where Satoshi Nakamoto would announce later Bitcoin [128]. Chaum's ideas are supposed to have set the technical roots of the vision of the Cypherpunk movement [129]. Chaum patented the blind-signature scheme preventing others from developing ecash system that use the same protocol. Violating Chaum's patents, Cypherpunks implemented an e-cash system called Magic Money [49] which further developed his technology.

However, DigiCash failed to gain public attraction. Its main problem was that banks and merchants would not adopt it for commercial and financial purposes.

In 1991, Haber et al. [87] proposed a scheme for *secure timestamping* of digital documents using digital signatures and hash pointers in previous documents, thus creating a chain of document certificates. In an improvement which was proposed later documents were collected into a block, all the blocks linked together in a chain. This data structure form of the skeleton for Bitcoin's blockchain.

A year later, cryptographers Dwork and Naor came up with a solution to email spamming [62] by using *computational puzzles*.They created the idea of *Proof-of-Work*. In 1997, Adam Back proposed a similar idea that he called Hashcash [15]. Back's Proof-of-Work system set the basis for Bitcoin's consensus mechanism.

Wei Dai at 1998 proposed b-money [50], an anonymous distributed electronic cash system, in which everyone could create money using a Proof-of-Work mechanism like Hashcash. B-money used the notion of a secured timestamp ledger as used by Haber et al. for digital documents [87].

In 2008, Satoshi Nakamoto published a paper under the title *"Bitcoin: A Peer-to-Peer Electronic Cash System"* on The Cryptography Mailing list at metzdowd.com [8] describing the Bitcoin protocol. The Bitcoin network came into existence on 3 January 2009 with the release of the first Bitcoin client, `wxBitcoin`, and the issuance of the first Bitcoins [9, 69]. Bitcoin represents a decade long work of research in cryptography combining several prior inventions [7].

Bitcoin is the first decentralized digital currency. The Bitcoin network is a fully distributed peer-to-peer network that anyone can freely join by running an open source implementation of the bitcoin protocol. Bitcoin does not rely on a trusted central authority and was he first applied solution to overcome the *Byzantine Generals' Problem* [113]. Bitcoin makes use a Proof-of-Work consensus mechanism that keeps the public ledger consistent, prevents double-spends and confirms transactions. It can also be used to achieve consensus on decentralized networks for elections, lotteries, asset registries, and more [7].

Running from 2009, Bitcoin is the most well studied Blockchain network [79] with various published papers on different topics such as privacy [32, 148, 149, 153], economics [14, 24, 37, 115], attacks [17, 70], network [59, 91] and scalability [46, 103, 104]. Over the years, Bitcoin's blockchain has grown significantly is size making it difficult for certain devices to store all of it and run as a full node. Specifically, In April 2018 the Bitcoin's blockchain size is over 160GB [28].

Bitcoin does not scale efficiently. There is a limit on the amount of transactions the bitcoin network can process. The maximum transaction processing capacity is estimated between 3.3 and 7 transactions per second [46]. The limit is related to the block size limit – the total number of transactions a block can contain – that was introduced as an anti-spam measure. Various solutions proposed to address this issue. One implemented solution is *Segregated Witness* (SegWit, BIP 141) [26]. The SegWit implementation increased the block size by a factor of approximately of two.

3.2 Identity

The identity of a blockchain user is defined as a cryptographic key pair (s_k, p_k), where s_k is the private key and p_k the public key. The private key is used for spending a "coin" and the public key is used as the receiving *address* of the user. No real-world name or identifying information are required [31, 128].

In Bitcoin, due to the public nature of the blockchain, it is possible to trace the flow of money between addresses and conclude that they are likely controlled by the same individual [31, 92]. For that reason, Bitcoin's identity system is considered *pseudonymous*. The underlaying non-anonymous Internet infrastructure – nodes leak their IP address when broadcasting transactions – together with the availability of all bitcoin transactions in the blockchain can be a threat to anonymity [31, 92, 124, 132, 146, 148]. Although Bitcoin let users to create new addresses at any time for any transaction, various techniques, such as transaction graph analysis, can be utilized to link together different addresses controlled by the same user [31, 124, 132, 146, 148].

Different cryptocurrencies, such as Zerocash [152] and CryptoNote [167], were built differently in ways that improve bitcoin's anonymity. In particular, Zerocash utilize zero-knowledge proofs (zkSNARKs [20]) which reveal no information at all about the amount or recipients enabling a completely untraceable ledger. On the other hand, CryptoNote uses *ring signatures* creating a *mixing protocol* satisfying both *untraceability* – for each incoming transaction all possible senders are probable – and *unlinkability*. CryptoNote compared to Zerocash has better performance but weaker anonymity [31].

3.3 Network

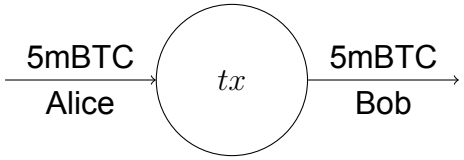
Blockchain networks are *peer-to-peer* networks. Anyone who wants to and spend and receive coins can freely join and participate in the network by running a software on their computer. All the nodes of the network are equal. There is no central server or trusted authority, neither hierarchy within the network. Both the protocol as well as the software are open. Every node connects to other peers of the network using peer-to-peer discovery schemes. In Bitcoin, there are some nodes called seed nodes, that their IP address is hardcoded in the code. This nodes can be used to quickly discover other nodes. Alternatively, a known IP address of a bitcoin node can be given manually.

3.4 Transactions

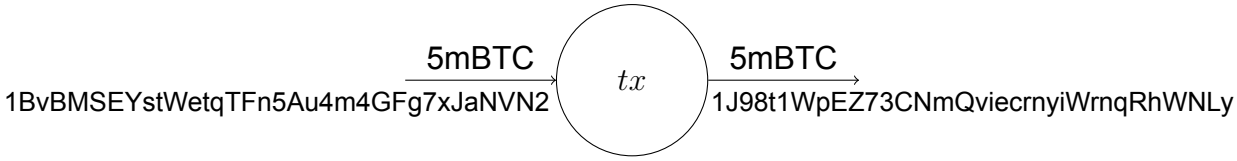
The basic data structure of the blockchain is *transaction (tx)*. A transaction transfers assets from one party to another. When a user wants to transfer coins to another user she must

sign a transaction. The signature certify that the sender owns the coins.

A blockchain transaction can be described as a *node* with two *edges*, one *incoming* edge and one *outcoming* edge [178]. The incoming edge describes the sender (from) and the outcoming edge the recipient (to). Each edge have each entity's address; its public key. Every bitcoin transaction has a unique *transaction id (txid)* which is derived by double hashing the transaction with the use of the SHA-256 cryptographic hash function. Payments are done by *linking* transaction nodes and money can be view as a chain of transactions where monetary value pass [178]. All transactions together form a transaction graph which is public. Every participant of the network can access and add a new transaction to the transaction graph.



(a) Simplified



(b) Addresses are public keys

Figure 12: A bitcoin transaction

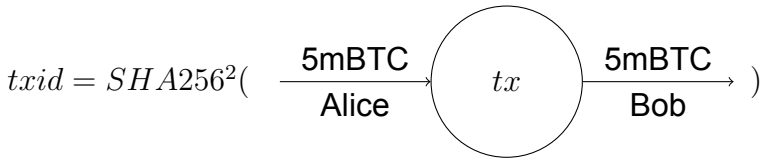


Figure 13: Bitcoin Transaction ID

When a transaction is occured, each node on the network should be able to confirm that the user making the transaction, has indeed the amount of transfer. Otherwise, anyone could produce money arbitrarily, sign it and create a valid transaction. As there is no central authority the whole network have to maintain exactly who has how much coin.

A transaction can have *outgoing unlinked edges*, edges that are not connected to another node transaction. These edges are unspent coins, owned by various users and ready to be spent. This type of edges are called *unspent transaction outputs* (UTXO). The UTXO set, the list of all outgoing unlinked edges, describes how much coins each user owns and is kept by every node of the network. In this way, the network can confirm, by checking the UTXO set, that the sender owns the money at the time of the transaction.

When a user wants to transfer money to another user, she have to find a previous transaction with a UTXO that she owns. Then she creates one transaction with one incoming and one outgoing edge and connects the incoming edge of the new transaction with the UTXO of the previous transaction. The previous UTXO is removed from the UTXO set. The outgoing edge of the new transaction is unconnected and becomes a new UTXO. The new UTXO specifies the value and the owner (address) of that edge. Finally she signs the transaction. The signing of the transaction is a way of declaring asset ownership. The

user have to prove that for a public key – the address where the UTXO points to – she holds the corresponding private key. Only the owner of the private key can produce valid signatures that are verifiable under the corresponding public key.

The UTXO set is maintained collectively by the network. When a full node is connected for the first time to the network, the nodes with which it connects inform it about the UTXO set and the history of all transactions that occurred from the begging of time. If a full node reconnects after a period on inactivity the node is informed about the new transactions that took place since the last time the node was connected to the network.

Each transaction is published on the network. This is achieved through a mechanism called broadcasting. When a transaction is created, the participants broadcast the details of the transaction to their neighbors. The neighbors publish the transaction to their neighbors and recursively the transaction is being published until the whole network becomes aware of it.

In Bitcoin a user cannot give changes. This is due to the lack of accounts and balances. The only way of keeping track money ownership is with the use of the UTXO set. For that reason, the total value of the income edge must be consumed at once. To overcome this problem bitcoin uses multiple incoming and outcoming edges for each transaction to create a changing system. In particular, when a user wants to transfer money to another user and the value of the UTXO edge is bigger than the desired amount, she can create a transaction with two outcoming edges, one for the recipient and one to give changes to herself.

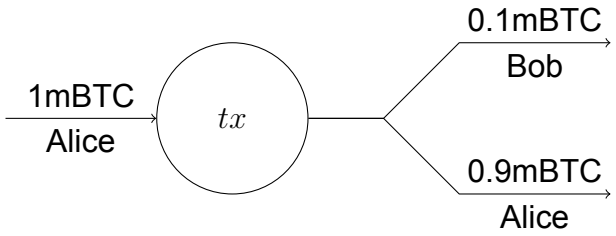


Figure 14: Change exchange

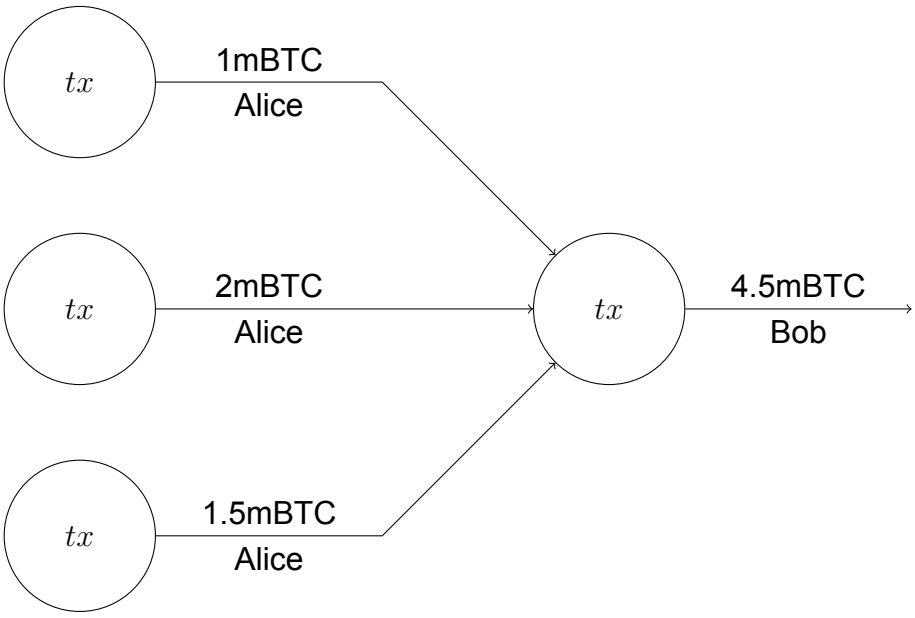


Figure 15: Transaction multiple inputs

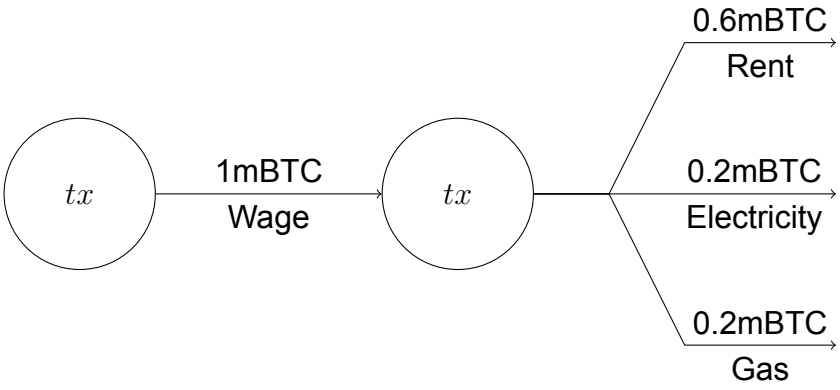


Figure 16: Transaction multiple outputs

In Bitcoin each transaction must satisfy the Kirchhoff's property. Kirchhoff's property mandates that the total outputs of a transaction are at most equal to the total inputs. In particular, let txs be all transactions of the network, $out(tx)$ all the output edges of the transaction tx, $in(tx)$ all the input edges of the transaction tx and $w(e)$ the value of that edge, it holds that:

$$\forall tx \in txs : \sum_{o \in out(tx)} w(o) \leq \sum_{i \in in(tx)} w(i)$$

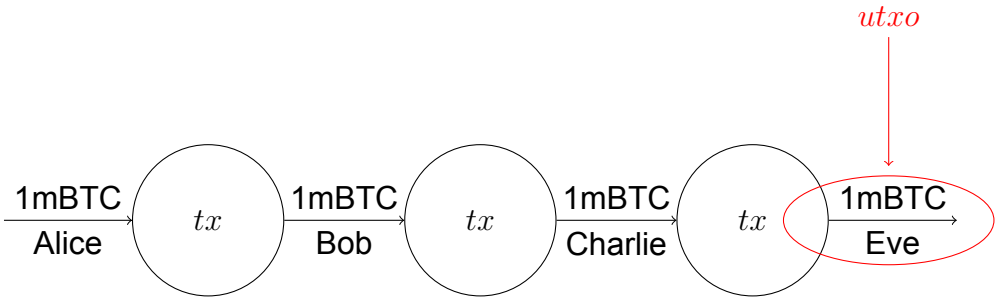


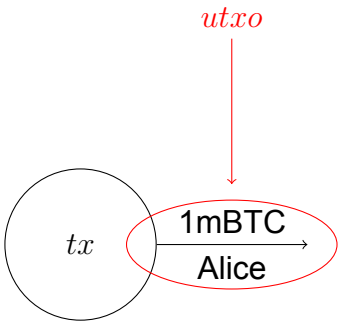
Figure 17: Transaction graph and UTXO

3.5 Blocks & Blockchain

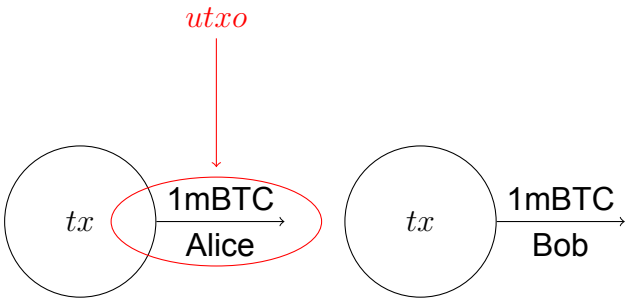
A double spent or a double spent attack is the action where a user sends the same transaction twice; transactions that spend the same UTXO. For example, Eve buys a coffee from Alice and creates a transaction that pays Alice. At the same time, Eve creates a transaction that sent the same amount to herself. Eve get her coffee and leave. Alice learn about the double spent later. Both transactions are valid: the signatures are valid and the Kirchhoff's property holds for both transactions.

As the network is decentralized and there is *latency*, a double spending may not be immediately noticed. In this case, it is impossible to tell which transaction occurred first and which second. To prevent double-spending the transaction must be put in *chronological order*. This way nodes of the network are sure if transaction A precedes transaction B. Furthermore, the order must be common for everyone.

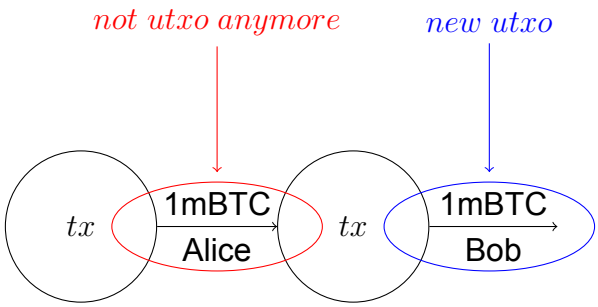
To achieve chronological order Bitcoin utilize a data structure called *blockchain*. The blockchain is an ordered back-linked list of *blocks*. Each block contains a set of trans-



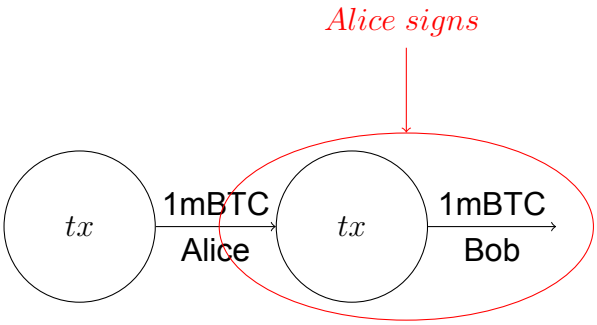
(a) Alice finds one UTXO that belongs to her



(b) Alice create a transaction with recipient Bob



(c) Alice connects the incoming edge of the new transaction with the old UTXO



(d) Alice signs the transaction. No one else can forge this signature

Figure 18: Speding money

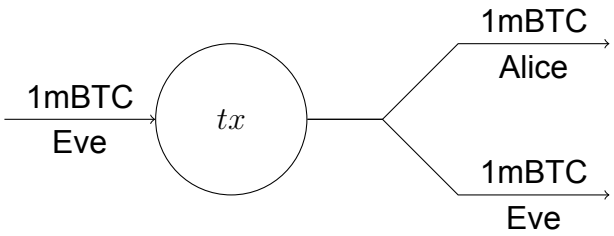
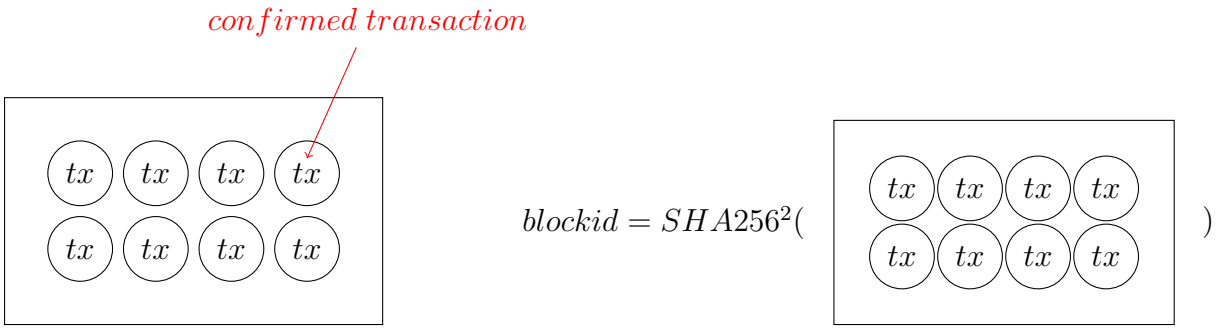


Figure 19: A double spent

actions. A block cannot contain double spends and each transaction can appear only once in a block. A transaction is called *confirmed* if it is in a valid block. A valid block cannot contain a transaction that spends an already spent UTXO by another transaction in a preceding block. Transaction A precedes transaction B if A is contained in a previous block from B. To ensure that a transaction is not a double spent, a user have to wait until the transaction contained in a valid block and thus confirmed. In the bitcoin network a block is set to be created approximately once every *ten minutes* and every newly created block contains the most recent transactions that did not exist in previous blocks.

Every bitcoin's block has a unique *block id* which is derived from the double hash of the header of the block with the use of the SHA-256 cryptographic hash function. Each block references to the previous block id known as the *parent block* through a pointer. Thus, every next block contains the hash of the previous block. This results in every next block in the chain requiring the previous block to have been already computed [178].



(a) A block **(b) Bitcoin Block ID**

Figure 20: Blocks

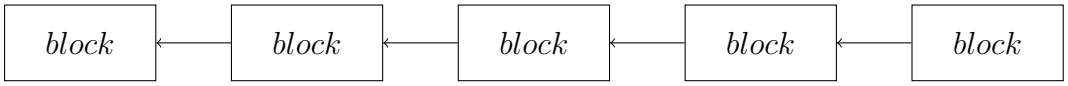


Figure 21: The blockchain

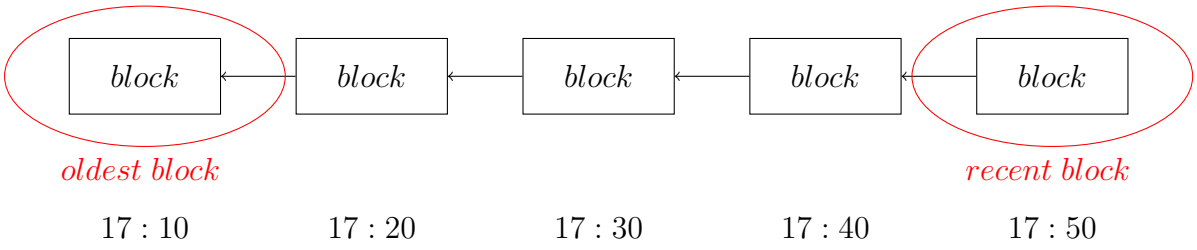


Figure 22: The blockchain timeline

3.6 Consensus mechanisms

The use of the blockchain data structure achieves transaction chronological order. What remains is a *global agreement* on the order of the blocks among the participants of the network. Someone can easily change the order of two blocks within the chain by changing the respective hashes or producing new ones wherever required [178]. This would allow an adversary to fake the order of transactions in time, which is an undesired outcome.

The global agreement on a common truth, the global blockchain (*ledger*), is called *consensus* – a single universal "truth". The consensus mechanism is the core mechanism of the blockchain. Through consensus, the *shared state* of the ledger comes to an agreement allowing all the nodes of the network to reach the same ledger state. Through consensus the users of the network agree on a common order of the blocks in the blockchain and therefore on the order of the transactions. Achieving consensus in a distributed system is challenging. A consensus mechanism has to be resilient to node failures, network delays and the existence of malicious nodes.

At high level, every public consensus mechanism works as follows: A "game", involving randomness, is taken place where each node of the network participates. The winner of the game is eligible to propose the new block that will be adopted in the blockchain. The probabilistic nature of the process is paramount to its security [79].

There are three basic consensus mechanism categories:

1. Proof-of-Work (PoW).

2. Proof-of-Stake (PoS).
3. Practical Byzantine Fault Tolerance (PBFT)

3.6.1 Proof of Work (PoW)

A Proof-of-Work (PoW) consensus mechanism, is a consensus mechanism where each node of the network tries to solve a computational puzzle that is computational *hard*, but feasible to find, and easy to verify *correctness*. Assuming that cryptographic hash functions are hard to invert, proof of work usually is established by seeking a range-collision of the hash function on the block [178].

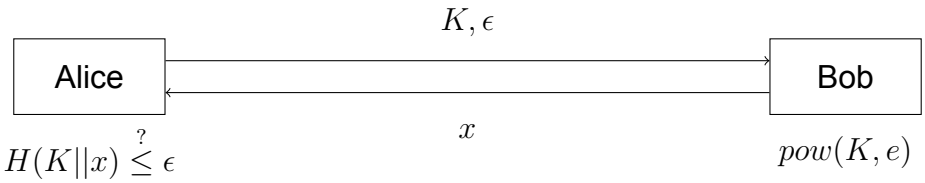


Figure 23: The Proof-of-Work protocol

Bitcoin [179] is the first blockchain system that uses Proof-of-Work for achieving consensus among the nodes of the network. A *target* ϵ is given and is asked that the hash of the block is *smaller* than the target. The node can only modify a *nonce*. By changing the nonce the node can change the block's hash (id). As cryptographic hash functions is assumed to be one-way, the only way to find a hash value smaller than the target is with a series of *brute-force* trials of different nonce values. Bitcoin's proof-of-work can be summarized as:

$$H(txs||nonce||parent\_blockid) \leq \epsilon$$

The network evaluates collectively the target value using a predefined algorithm. Thus, the difficulty of proof-of-work and the frequency with which the blocks are generated are controlled by the network. In Bitcoin the expected *block generation rate* is one block per 10 minutes.

All the nodes of the network simultaneously try to produce a new block, meaning they try to find a correct nonce satisfying the proof-of-work requirements. The block the node produces includes all the valid transactions – transactions that are not in a previous block – and a reference to a parent block. Each block has to meet the target proof-of-work requirements. If not the block is considered invalid.

When a node finds a hash value less than the target, it gets to add the proposed block to the blockchain. It broadcasts the new block to all its neighbors which, in turn, transmit it to the whole network. When a block is found by another node, all the nodes stop the block production and start over upon the new block. A block is valid if it contains valid transactions, a valid proof-of-work and a reference to a known valid parent block.

The existence of a transaction in a block makes the transaction valid. The deeper the block is in the blockchain the more difficult is for an adversary to alter the block. The reason is that the time an adversary needs to alter a block grows *exponentially* in the number of blocks that have followed [79] – she will need to reproduced those blocks and the blockchain is constantly extended. Waiting 6 confirmation blocks to appear after the

block in which the transaction is confirmed is enough for a transaction to be considered secure.

A PoW system is based on *randomness*. Each node has a small change to win the block which is approximated proportionally to the computational power of each node. PoW depends on having a majority of the miners acting honestly out of self-interest [7].

PoW consensus mechanism works very well in *public* blockchain systems where trust of the nodes is low. It eliminates the double-spend problem and guards against *Sybil attacks* [60]. However, the transaction confirmation time is longer compared to conventional financial services (such as VISA) [57, 158, 179] resulting in slower transaction confirmation rates. Lastly, the energy waste attributed to the mining process can be very high – the energy requirements of the Bitcoin protocol are estimated to be comparable to those of a small country [133].

```
x = rand();

do {
  ++x;
} while (H(K||x) >= ϵ);

return x;
```

Code 1: A simple Proof-of-Work Algorithm

3.6.2 Proof of Stake (PoS)

Proof-of-Stake algorithms are designed to overcome the disadvantages of PoW in terms of the high electricity consumption involved in block generation [18] and provide equal security guarantees [105]. Unlike PoW where the nodes of the network solve computational puzzles in order to create a new block, in PoS the choice of the block creator among the miners is random, yet relative to the *stake* the node possesses according to the current ledger. Maintaining the blockchain relies on the *stakeholders* themselves and assigns work to them based on the amount of stake that each possesses as reported in the ledger [105]. The higher the stake participant, the higher the possibility to be chosen.

Proof-of-Stake algorithms suffer from the so-called "*nothing at stake*" problem. The "nothing at stake" problem refers to attacks against PoS blockchain systems where shareholders do not have *incentives* to follow the protocol and vote simultaneously on multiple blockchains exploiting the fact that little computational effort is needed to build a PoS blockchain [105]. Blockchains with PoS as a consensus mechanism can be either *permissioned* or *permissionless* in the sense of stake availability rather than node authorization. A node to participate in block election has to possess a stake. If the market where the stake is available for sale is public and accessible for anyone then the blockchain is considered as truly permissionless. Otherwise, the initial stakeholders can sell stake selectively to participants of their choice making it permissioned. Ouroboros is the first *provable* secure PoS algorithm [105] and is the main consensus algorithm of the Cardano blockchain [71].

3.6.3 Practical Byzantine Fault Tolerance (PBFT)

The *Practical Byzantine Fault Tolerance* algorithm (PBFT) [38] is a high-performance Byzantine Fault Tolerance [113] consensus mechanism. It is based on the concept of

state machine replication and *replication state voting*, and is able to process tens of thousands of requests per second with minimal latency. PBFT has only a 3% overhead over a typical filesystem [38].

PBFT and state-machine replication protocols' downside is poor *scalability*. The number of nodes (replicas) [169] that can be supported is very limited. PBFT has only been scaled and studied up to 20 replicas [18, 169]. To overcome this limitation, without compromising security, various PBFT variants, such as Ripple and Stellar, partition the network into smaller groups called federates and each one runs a local consensus protocol among its members. A global consensus is achieved when certain conditions are being met [57].

Table 1: Blockchain consensus mechanisms. Adapted and modified from [18, 169]

	PoW	PoS	PBFT
Blockchain Type	Permissionless	Both	Permissioned
Scalability of nodes	High	High	Low
Scalability of clients	High	High	High
Transaction rate	Low	High	High
Latency	High	Low	Minimal
Power consumption	High	Low	Low
Token needed	Yes	Yes	No
Cost of participation	Yes	Yes	No

3.7 Mining & incentives

The process of creating a block in a PoW consensus system is known as *mining* and the participants as *miners*. Miners contribute their computation resources to validate and generate blocks. This process can be costly. For that reason incentives motivates the miners of the network to mine blocks. In permissionless blockchains such as Bitcoin or Ethereum a monetary incentive in the form of cryptocurrency incentivize the miners while in permissioned blockchains incentives can be financial or acquiring access to valuable information [107].

In Bitcoin a reward is given to the miner who its proposed block got into the blockchain. This happens through a specific type of transaction called *coinbase*. This type of transaction has an unconnected income edge without a sender and an outcoming edge with recipient the miner that produced the particular block. This is how new bitcoins are generated. The amount of coin mined in each block is pre agreed by the network and every four years is reduced by half. At the moment of writing, the reward is 12,5 BTC.

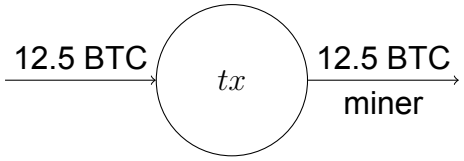


Figure 24: Coinbase transaction

Another way a miner is rewarded is by collecting *transactions fees*. A transaction fee is the difference between the total value of all incoming edges and the total value of all outcoming edges. In particular, the total fees a miner collects by a block are defined as:

$$fees = \sum_{tx \in block} [\sum_{i \in in(tx)} w(i) - \sum_{o \in out(tx)} w(o)]$$

3.8 Blockchain fork

Due to network latency and the distributed nature of the system, it is possible that two miners will find a block around the same time and some nodes will accept the first block and some others the second one. Both blocks are valid, containing a valid proof of work, valid transactions and both extent the same parent block. In such cases there is a temporary *fork* in the blockchain, where some nodes are adding blocks to one branch while other nodes are adding blocks to another branch. As a result, two competitive versions of the blockchain will emerge [7]. Similarly to transactions the order of the blocks cannot be decided.

To resolve this, each node always selects the longest brach to extent. At some point, one of the two branches will be extended by a new block. The nodes that were working on the first branch will see that the new branch is the longest and start working immediately on that. Eventually the system will come to an agreement, the longest branch will be accepted and other blocks will be discarded. The process of chain selection can be likened to voting where mining nodes "*vote*" with their mining power by choosing which chain to extend by mining the next block; the new block itself represents the voting result [7].

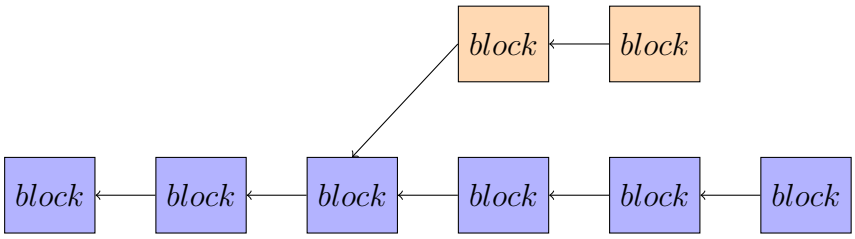


Figure 25: A blockchain fork

If a malicious adversary wants to double spent or execute a denial-of-service, she has to produce a malicious blockchain *longer* than the honest. To achieve that, an adversary would need to control the majority of the CPU power of the network and specifically more than 51% of the total network's hashing power. This is called the *51% attack*. It is important to note that this kind of consensus attack affects at best the most recent blocks. Beyond a certain depth blockchain is absolute immutable. In addition, this type of attacks cannot steal or spend coins. The adversary cannot forge a signature to produce a valid transaction impersonating another user. Bitcoin's security has been formalized and rigorously explored [79] over the last years.

Ethereum is designed to produce blocks very fast (around 12-15 second) in comparison to Bitcoin (around 10 minutes). Blockchains with fast block confirmation times, suffer from reduced security due to high *stale rate* [36]. To counterpart that, Ethereum use a variant of the *GHOST* protocol [158]. The GHOST protocol rule picks the chain that has had the *most computation* done upon it and not the chain with the longest depth.

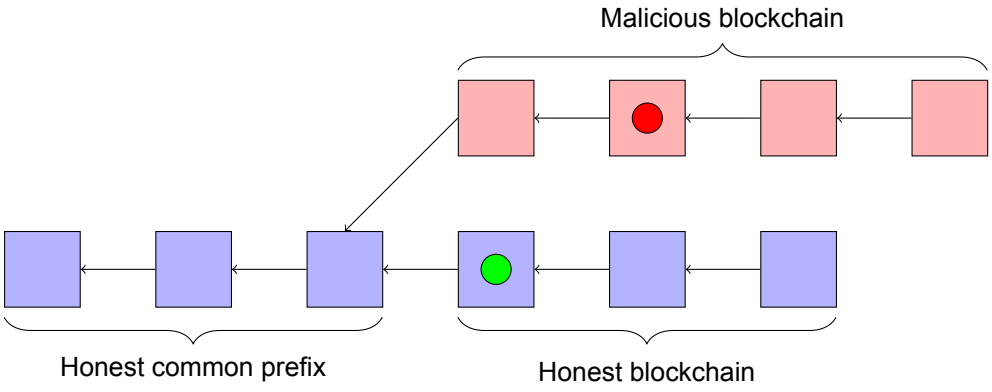


Figure 26: Double spent attack

3.9 Blockchain Types

There are various types of blockchains varying in restrictions on data access and participation in the consensus process. Each one has its own advantages and disadvantages.

- **Public Blockchain**: A public blockchain is a blockchain, in which there are no restrictions on reading blockchain data – encrypted or not – and validating transactions [86]. The most common implementation of public blockchain is Bitcoin [128] and Ethereum [72].
- **Federated or Consortium blockchain**: In a federated blockchain transaction validation is limited to a predefined list of entities with known identities to the network. Data access can either be public or restricted [86].
- **Private blockchain**: A private blockchain is a blockchain where consensus mechanism is centralized to one single entity regardless of data access [86].

3.10 Consensus defined types of Blockchain

- **Permissionless blockchain**: A permissionless blockchain is a blockchain, in which there are no restrictions on participation to the network [86].
- **Permissioned blockchain**: A permissioned blockchain is a blockchain, in which transaction processing is performed by a predefined list of subjects with known identities [86].

Table 2: Blockchain Types. Source [93]

	Public	Permissioned (Multiple Entities)	Private (Single Entity)
Participants	Permissionless Anonymous	Permissioned Identified Trusted	Permissioned Identified Trusted
Data Access	Public	Public or Restricted	Restricted
Consensus	PoW, PoS	FBTA, PoS	FBTA

3.11 Smart Contracts

The idea of *smart contracts* proposed in the early 1990s [163], by Nick Szabo. He defined them as a computer protocol intended to facilitate, verify, or enforce the negotiation or performance of a contract [162, 163]. They have been used primarily in association with cryptocurrencies enabling parties to formally specify a cryptographically enforceable agreements [31]. A smart contract consists of a set of promises including protocols within which the parties perform on these promises. It can define rules and penalties around an agreement and automatically enforce those obligations. The contractual rules may be partially or fully self-executed, self-enforcing or both. Regarding blockchain a smart contract is any computer program that is executed on the blockchain – a general purpose computation.

3.11.1 Bitcoin scripts

Bitcoin is the first blockchain that implements smart contract functionality. It provides a *scripting* language for expressing simple smart contracts such as ownership of an amount of coins by one or multiple entities. Bitcoin's language is a *stack-based* scripting language inspired by Forth [144]. It offers a set of simple *serial commands* supporting cryptographic primitives such as hash functions and signature verification. The main disadvantage of Bitcoin's language is that is not *Turing-complete* limiting the type of smart contracts one can create. Furthermore, adding new commands to extent functionality requires either a soft-fork which implements the new functionality or the creation of a blockchain anew. A soft fork has to be decided by the majority of Bitcoin's network. There is no guarantee that the nework will adopt the new change. On the other hand, the implementation of a new blockchain has the disadvantage that Bitcoin's mining power is lost. Numerous previous smart contract application atop Bitcoin (e.g., lottery [6, 23], verifiable computation [109]) have demonstrated the difficulty of Bitcoin's scripting language [106].

In 3.4, a transaction is defined as a graph node with two edges, one incoming and one outcoming and each edge contains an address. In reality, each edge contains a program which decides whether the edge can be spent or not. The program is written in Bitcoin's scripting language and is called `scriptPubKey`. This allows the expression of more complicated ownership of assets such as *multi-signatures* or *micropayments*. The script is executed on a stack machine and containes a series of simple serial commands. When a UTXO is spent, every node of the network executes the *script*. If the output of the program is 1, then the transaction is valid and can be spent. Otherwise, the transaction is considered invalid.

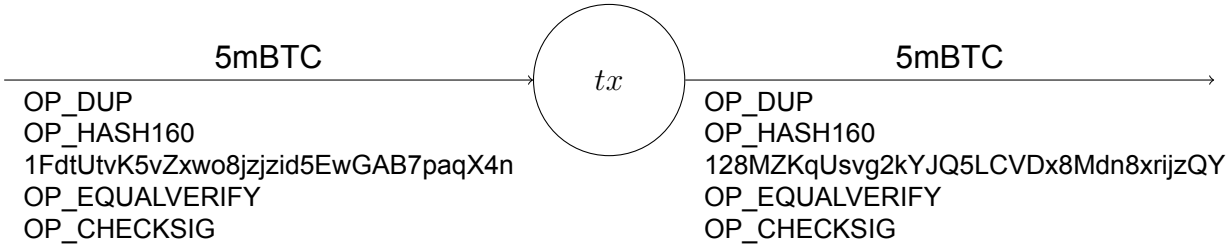


Figure 27: A Bitcoin script

3.11.2 Ethereum smart contracts

The need of user defined open source blockchain applications have been emerged. As a result, *Ethereum* [36, 174] was born. Ethereum is an open-source, public, blockchain-based distributed computing platform and a smart contract framework that enables anyone to build *distributed applications*. It provides a decentralized Turing-complete virtual machine, the *Ethereum Virtual Machine* (EVM) in which smart contracts are executed by all nodes of the network. In Ethereum, one can create smart contracts in a high level language, such as *Solidity* [74], which in turn is compiled to *bytecode* that is executable on the EVM. Ethereum provides a cryptocurrency called *Ether* which can be transferred between accounts, and *gas*, an internal pricing mechanism used to execute contracts and allocate resources on the network. Like Bitcoin, Ethereum use a Proof-of-Work consensus mechanism called *Ethash* and has been designed to be *ASIC-resistant* [72]. Soon Ethereum will be moved to a Proof-of-Stake consensus mechanism called *Casper*.

With Ethereum launch, the notion of *DApps* (decentralized applications) arisen and a lot of developers and companies are building numerous applications atop Ethereum such as prediction markets [11, 81], social media platforms [5, 16], online gambling [43, 68] and video games [47]. As of January 2018, there are more than 250 live DApps.

Before describing how smart contracts are deployed in Ethereum, a thorough analysis of Ethereum's core mechanisms must be made. Ethereum constitute of a set of one global object, that of *account*. Each account has a *state*, in which the nodes of the network must agree upon with the use of a consensus mechanism. A 20-byte address is associated for each account. The state of an account can only be changed by a transaction. The global state of Ethereum is made up of accounts. For that reason, Ethereum is often described as a *state machine* [36] where the *state transition function* takes as input a transaction and outputs a different state.

Each account is composed by the following fields: *address*, *balance*, and *nonce*. The address is a 160-bit account identifier derived from a public key. The balance contains the total amount the account holds in *Wei*; the smallest denomination of ether. Ethereum instead of using UTXOs to keep track of coin ownership, adopted the traditional notion of balances as in financial systems. The choice of balances over UTXOs was a design decision by the Ethereum foundation. Each option has its advantages and disadvantage. According to Ethereum founders, the advantages of accounts massively outweigh the alternatives [73]. The nonce is the total number of transactions sent from the account. The nonce is used as a *replay-attack* prevention mechanism. In the absence of nonce, a malicious adversary may send the same transaction twice and amount of the transaction will be deducted twice from the user's balance.

Similar to Bitcoin, anyone can send ethers from one account to another. The principles of the transaction described in 3.4 are also applicable to Ethereum. Each transaction consist of the following fields: *from*, *signature*, to, and *amount*. The from and to fields describe the sender and the receiver respectively. The signature field contains the valid signature of the transaction where the nodes of the network verify. Lastly, the amount contains the amount to be sent in Wei.

There are two type of accounts: *personal* (or external own accounts) and *contract* accounts. A contact account is an account dedicated for smart contracts. They contain two extra fields: *code* and *storage*. The code field contains the code of the smart contract and the storage field the internal persistent storage of the smart contract. Those fields are empty, and optional, when it comes to personal accounts.

Figure 28: An Ethereum account

Transactions contain an extra field, called *data* field. To create a contact, a user has to sent a transaction with empty recipient and data field the code of the smart contract. When the nodes of the network hear a transaction, where the field of the recipient is empty, they treat the transaction as a smart contract creation transaction. They create a contract account with code, the string that is contained in the data field of the transaction, and empty storage. After the creation of the contract account an address is returned to the creator of the contract. The address is created from the concatenation of the creator's public key and her account current nonce. Anyone knowing the address of the contract can interact with it. In particular, to call a smart contract function, the interested party has to send a transaction with recipient the smart contract's address and data the function's name along with its arguments.

Table 3: Ethereum accounts

	Personal account	Contract account
address	$H(p_k)$	$H(creator, nonce)$
code	$\emptyset$	Code to be executed
storage	$\emptyset$	Data of the contract
balance	ETH	
nonce	# transaction sent	

When a contract account is activated the contract's code runs. The contract can read or write to internal storage, do various computations, send messages or create new contracts. A contract account can not initiate new transactions on its own but only in response to a transaction initiated by a personal account – only personal accounts can change the state of the accounts.

Contract accounts can send *messages* to other contracts accounts. Messages are like transactions except it is produced by a contract and exist only in the Ethereum's execution environment. The network never sees the messages neither are stored in the blockchain. This way, contracts can have relationships with other contracts.

Table 4: Ethereum transactions

	create	send	call
from	creator	sender	caller
signature	σ	σ	σ
to	$\emptyset$	receiver	contract
amount	ETH		
data	code	$\emptyset$	$f, args$

The code of a smart contract is run by all nodes of the network. The code can change the state of the contract or the state of another contract or personal account. The nodes must agree on a global common state, the state of each account. Similar to Bitcoin, the transactions, that change the state of each account, are contained in a block. A proof-of-work consensus mechanism is used by the network to agree upon the block that will extent the blockchain. The new block contains the new state of all accounts.

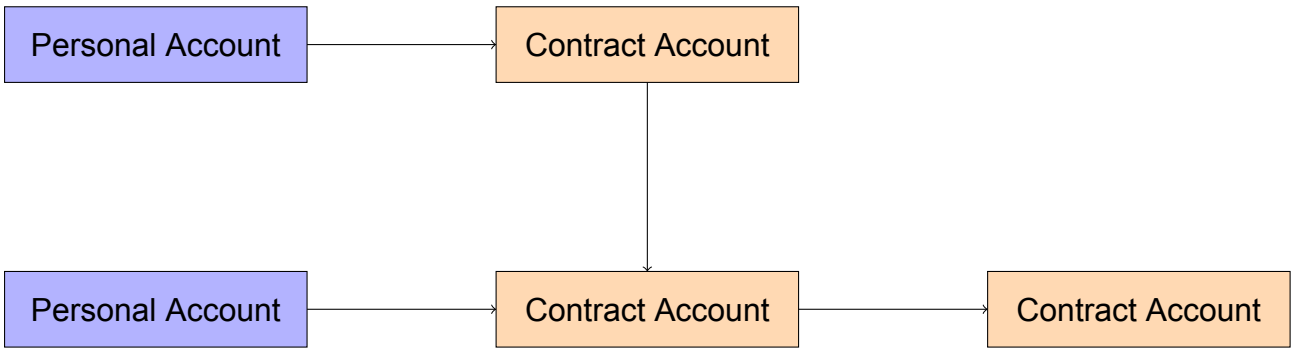


Figure 29: Transactions & messages

All nodes evaluate all transactions, execute code and store all state. Because Ethereum is turing-complete, infinite loops or computational heavy code could make all the nodes of the system to be occupied for a very long period of time, or even infinitely. Furthermore, storage requirements can be grown very fast making unable to fulfil and maintain that demand. To address this issues, Ethreuem use a unit called *gas* as a measurement of computation usage. It acts as an anti-denial of service mechanism. The intent of gas, is to enforce a malicious adversary to pay proportionately for every resource that consumes, including computation, bandwidth and storage [36].

Each gas unit has a gas price that is expressed in *gwei* (1×10^9 Wei). Every node in the Ethereum network executes instructions (*opcodes*), that represent the code of the contract, within the Ethereum Virtual Machine (EVM). Each of these instructions has an associated cost in gas. The cumulative sum of all the operations is the total gas cost for a transaction and is the fee that is being paid to the miners.

The cost of each opcode (operation code) of EVM is defined in Ethereum's yellow paper [174] and a gas cost summarization of basic opcodes are be shown in Table 5. For storing data, Ethereum offers two opcodes: the *SLOAD* opcode which loads a word from the storage and the *SSTORE* opcode which saves a word to storage. The size of an EVM word is 32 bytes (256 bit) and to store one EVM word using the SSTORE opcode costs 20.000 gas. Due to that reasons, storing data is expensive and for the moment blockchain technologies cannot be used for data storage or data processing.

An Ethereum transaction has two extra fields: the *start gas* and *gas price*. The start gas is the maximum amount of gas willing to pay and the gas price is the price willing to pay per gas unit. The start gas (or gas limit) acts as a protection to computational wastage or malicious code. Even if the code needs more gas to terminate, when the gas limit is reached the execution is stoped and all state changes are revert. An extra benefit of gas limit is, that a miner seeing the gas limit field can estimate the needed computational time beforehand and act accordingly. The gas price is not defined by any central trusted authority but is regulated by the network itself. By setting the gas price, the user in a sense "votes" for the value of the gas. Gas price determines how quickly a transaction will be mined; the higher the price is, the more likely is the transaction to be in the next block.

Figure 30: An Ethereum transaction

The start gas can be larger than the cost of the computation. In this case, all unused gas is refunded at the end of the transaction to the sender. If the computation exceed the

Table 5: Ethereum opcodes gas costs

Operation	Gas	Description
ADD/SUB	3	Arithmetic operation
MUL/DIV	5	Arithmetic operation
ADDMOD/MULMOD	8	Arithmetic operation
AND/OR/XOR	3	Bitwise logic operation
LT/GT/SLT/SGT/EQ	3	Comparison operation
POP	2	Stack operation
PUSH/DUP/SWAP	3	Stack operation
MLOAD/MSTORE	3	Memory operation
JUMP	8	Unconditional jump
JUMPI	10	Conditional jump
SLOAD	200	Storage operation
SSTORE	5.000 / 20.000	Storage operation
BALANCE	400	Get balance of an account
CREATE	32.000	Create a new account using CREATE
CALL	25.000	Create a new account using CALL
LOG	375	Logging operation

gas limit then a *out of gas exception* is triggered and the state is reverted to the previous one. Out of gas exceptions are not refundable. The miner still claims the fee for each computational step performed.

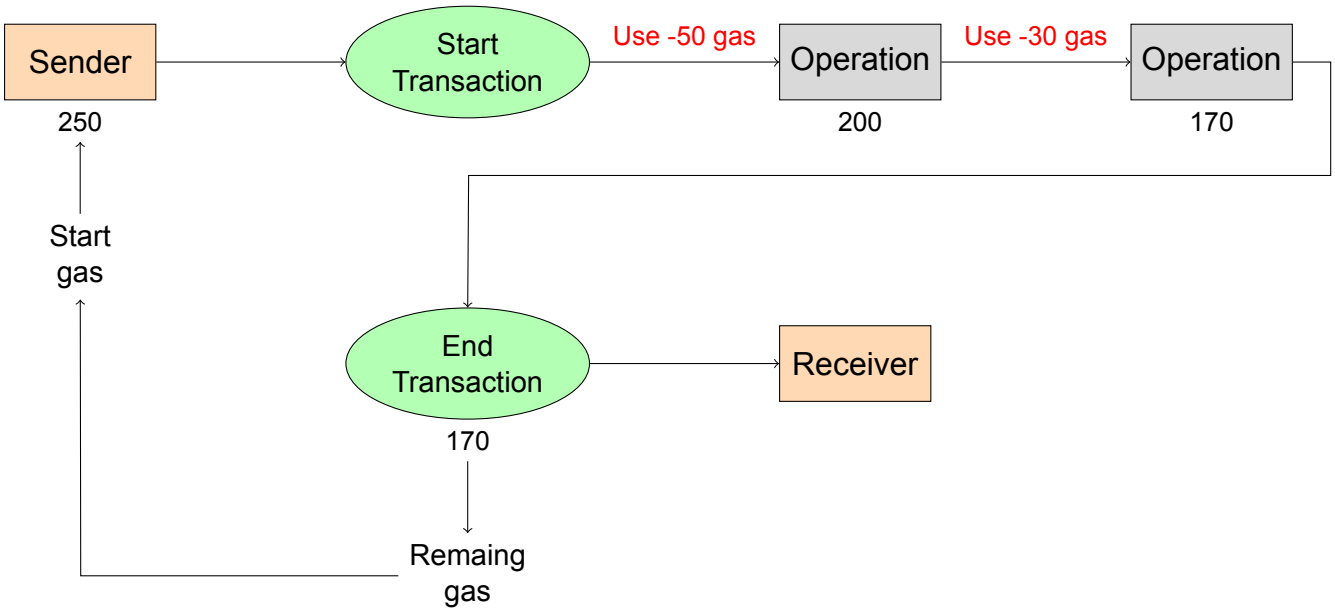


Figure 31: Smart contract execution

Each miner at code execution perform the following steps:

1. If $start\_gas * gas\_price > balance$ then halt
2. Deduct $start\_gas * gas\_price$ from $balance$
3. Set $gas = start\_gas$
4. Run code deducting from gas

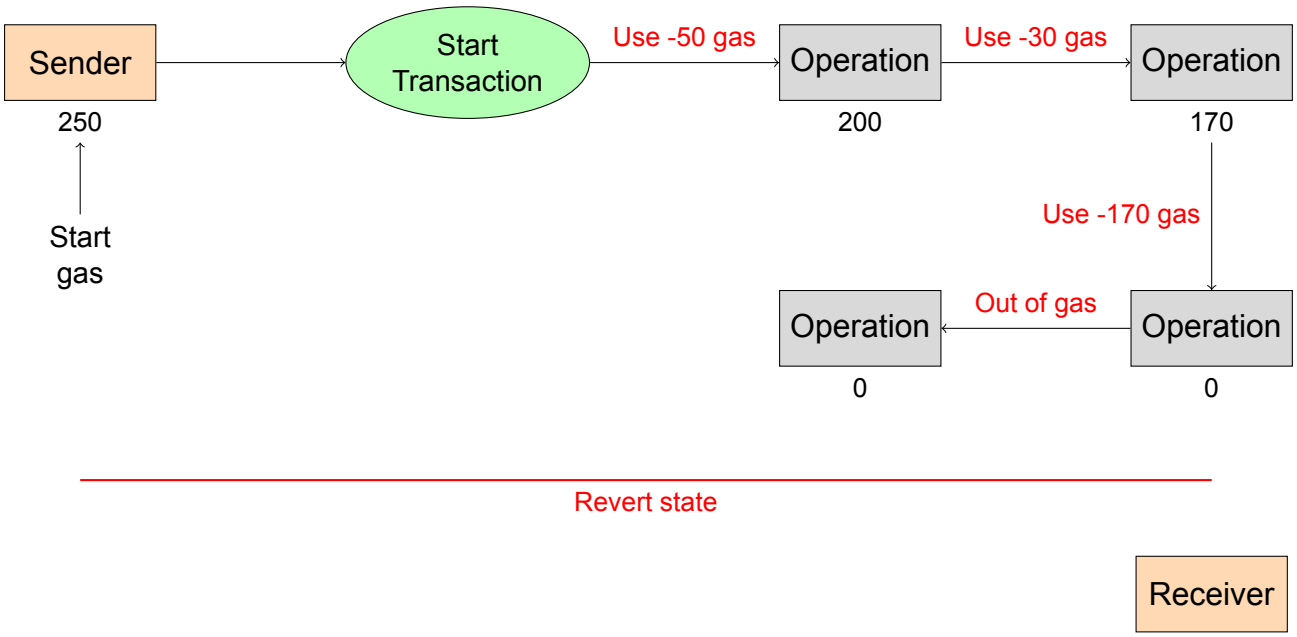


Figure 32: Out of gas exception

5. After termination return remaining gas to $balance$

The total gas cost of the transaction is paid to miner that mined the block. The *maximum fee* a miner could take by a transaction is calculated as:

$$fee_{max} = startgas \times gasprice$$

For example if the start gas is set to 50.000 and the gas price to 20 Gwei the maximum fee is 0.001 ETH.

As discussed earlier, the code of the contract is executed within the Ethereum Virtual Machine (EVM). EVM is a stack-based machine and uses 32-bytes (256-bit) words. It reads a series of bytecode instructions (EVM code) where each bytecode represents an operation (opcode). Lastly, it provides various built-in cryptographic primitives.

```
PUSH1 0
CALLDATALOAD
SLOAD
NOT
PUSH1 9
JUMPI
STOP
JUMPDEST
PUSH1 32
CALLDATALOAD
PUSH1 0
CALLDATALOAD
SSTORE
```

Code 2: EVM bytecode

Writing smart contract on bytecode can be challenging. For that reason, various high level programming language that compile to EVM code have been implemented. The most wide used is Solidity [74]. From the perspective of a developer, Solidity is much alike to JavaScript as it supports most of its structures. Every contact have to be declared

with the keyword `contract` at the begging of the file. It is the same as declaring a class or an object in object-oriented programming languages.

Solidity supports two types of variables: *state* variables and *local* variables. State variables are contract variables that are permanently stored in contract storage and have to be declared at compilation time. Local variables are function variables that cannot be accessed outside the scope of the function and they can be stored either in storage or in memory.

Solidity supports the following variable value types:

- Boolean: true of false.
- Integers: Signed and unsigned integers of various sizes. Keywords `uint8` to `uint256` in steps of 8 (unsigned of 8 up to 256 bits) and `int8` to `int256`.
- Fixed-size byte arrays: `bytes1` to `bytes32` in steps of one.
- Dynamically-sized byte array: `bytes` or `string`.
- Address: 20-byte value holding an Ethereum address.
- Enum: Enumerated type.
- Mapping: A key-map type similar to hash tables. `mapping(keyType => valueType)`.
- Structs: A structure. It can be used to define new types.
- Function: Variables that hold a function reference.

EVM supports a built-in logging mechanism which can be used by smart contracts to notify for various events. Services outside the blockchain can register listeners to this events and act accordingly. In solidity, an event is declared by the `event` keyword along with its arguments.

Lastly, Solidity supports multiple inheritance including abstract class and interfaces. Contracts can extend another contract with the keyword `is` and access internal functions and non-private members.

Since smart contracts deal directly with currency exchange, security of smart contract is of the utmost importance [54, 118]. The DAO bug [1, 51] is a perfect example. At least 60 millions US dollars was lost leading to a hard fork in Ethereum and the creation of Ethereum Classic.

In contrast to traditional application that can be patched when bugs are detect, smart contracts, due to the nature of blockchain, are irreversible and immutable. Various analysis [54, 118] have shown that most of the deployed smart contracts on Ethereum are vulnerable. In particular, an analysis tool called `Oyente` [118], created by Loi Luu et. al, marked 8,833 out of 19,336 smart contracts as vulnerable. Is believed that these bugs arise from the gap in understanding the actual mechanisms of the underlying platform of Ethereum [118]; developers make false assumptions of the semantics of the system.

3.11.3 Cardano

Another platform for smart contract implementation is Cardano. Cardano is a security focused blockchain that utilize the latest research and engineering insights to build a platform suitable for the highest value applications [71]. It supports distributed applications creation and smart contracts verifiable by a method called formal verification allowing logical proof of correctness of code providing high security. Cardano addresses the need for regulatory oversight while maintaining consumer privacy and security. Cardano is the first blockchain project to be peer reviewed by academic researchers [71] and its consensus mechanism, *Ouroboros*, is the first Proof of Stake algorithm to be provably secure [105]. Cardano consists of two main layers, one for accounting and one for computation. The accounting layer is called *Cardano Settlement Layer* (CSL) and the computation layer *Cardano Computation Layer* (CCP) where distributed application can be built and run upon. The CCP layer has not been implemented yet and there is a plan to be released as a beta by the first quarter of 2018 [139].

```
pragma solidity ^0.4.16;

contract Namespace{

  struct NameEntry {
      address owner;
  }

  uint32 constant REGISTRATION_COST = 100;
  uint32 constant UPDATE_COST = 10;
  mapping(bytes32 => NameEntry) data;

  function nameNew(bytes32 hash){
      if (msg.value >= REGISTRATION_COST){
          data[hash].owner = msg.sender;
      }
  }

  function nameUpdate(bytes32 name, bytes32 newValue, address newOwner){
      bytes32 hash = sha3(name);
      if (data[hash].owner == msg.sender && msg.value >= UPDATE_COST) {
          data[hash].value = newValue;
          if(newOwner != 0) {
              data[hash].owner = newOwner;
          }
      }
  }

  function nameLookup (bytes32 name) {
      return data[sha3(name)]
  }
}
```

Code 3: An Ethereum Smart Contract

4. PROBLEM STATEMENT

4.1 Overview

The continuous discussions about Big Data over the past few years has drawn the attention of researchers and individuals who try to understand their complex meaning and effects in people's lives. The new Data Age we live in has surely caused great shifts in pre-existing understandings of the world. It has transformed the way information is produced, transmitted, processed and stored, thus altering its value as well as the whole mode of operation of the global markets.

The evolution of computer power processing, hardware capacity and, later, cloud storage contributed to the creation of information economy. Technological advancements led to the creation of handheld devices able to run multiple applications, use sensors, and produce, transmit and store huge amounts of data. The amount of existing data is rapidly increasing and it is estimated that 20% of the world's data have been collected over the past few years [156, 181]. Commercial use of data is only one of the many and diverse industries that big data have decisively influenced. Some examples of data intensive fields are marketing and advertising, healthcare, transportation, energy regulation and distribution, retail and demographics, and sensors embedded in products for control, or IoT [108].

The value of raw data varies from a hundred cents to over several hundred dollars per individual [48, 63, 66]. The more it is being analyzed, the more its value increases. As some put it [10, 53, 55, 89, 110, 123, 123, 136, 159, 166]: data is the new oil. And like with all sources of wealth, they are related to multiple interests and, thus they should be given the attention required.

Personal Data

People constantly produce and publish data about themselves leaving a constant moving digital fingerprint all over the Internet; their personal data. The type, quantity and value of personal data being collected are vast [155]: bank accounts, medical records, employment data, web searches, sites visited, likes and dislikes, product purchase histories, quotes, tweets, texts, emails, phone calls, photos, videos, personal moments, emotions as well as coordinates of real-world locations.

Personal data can be gathered in three ways [155]:

- Volunteered data: data shared explicitly by the individuals
- Observed data: data captured by recording actions of individuals
- Inferred data: data based on analysis of individuals

Big data cannot exist without personal input and user cooperation. Companies create free services in return for user input leading to consumer-driven big data collection.

Personal data are typically gathered by a few big tech companies that own and privatize them. This happens as personal data belong to the institutions that collects them. This condition gave rise to the data broker and mining industry creating data marketplaces [117,

141, 161] and trade platforms where data can be sold and bought for a price. All data is potentially for sale without the consent of the rightful owners, the users.

The existing centralized models in which third-parties collect and control vast amounts of personal data can be questionable. Individuals have little or no control over their personal data and what these data are used for. Public concern about user privacy and protection is raised as related new research is being published [181]. When personal data are not controlled by their producers, the rightful owners, individuals are not protected against market and state policies. The invasion of privacy is a severe offense that derives from unregulated data exploitation. All users connected to platforms and data harvesting applications should be empowered with the ability to own their personal data, to control how data are being collected, used, shared and by whom.

Open Data

Nevertheless, the ability to have access and process large volumes of information can still be beneficial for individuals and the societies at large. Creating Open Data is an important goal of the research community and has a lot of advantages. By allowing researchers to perform their research on datasets referring to public data records accountability, fault detection and trust in the validity of prior research can be achieved [64]. The Open Data Movement [172] is a paradigm of free data distribution without limits, copyrights and patents. However, there is a growing public concern about privacy preservation of research participants, which can become subversive for data collection. As legislation in the form of privacy disclosure acts is forced, data become private, sensitive and protected. At the same time though, data processing is rendered impossible, inefficient or false.

4.2 Regulations

The EU Data Protection Directive 95/46/EC (DPR) [134] defines personal data as follows:

> personal data shall mean any information relating to an identified or identifiable natural person (’data subject’); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity;

However there has been a clear notion that the data subject can potentially be identified by pseudo-identifiers [121]. In the new General Data Protection Regulation (GDPR) [135] forced by the EU this has been formalized as:

> a data subject is one who can be identified, directly or indirectly, by means reasonably likely to be used by the controller or by any other natural or legal person

The recent approval of GDPR [135] in 2016 by the European Commission (EC) imposes new obligations on data controllers and processors in contrast to the previously adopted Data Protection Directive (DPD) [134]. New legislation aims in the extension of responsibility and accountability requirements of organizations also demanding explicit consent of the data subject (person) while securing her right to withdraw, and to be forgotten [131].

Key changes of the new data protection law introduced in GDPR in contrast to the DPR are presented below [131]:

- organizations based outside EU that process personal data of EU residents are bound by the new legislation
- All EU member states are obliged to comply to a single set of rules
- The responsibility and accountability requirements of organizations are extending
- EU residents are empowered with explicit consent over their data, maintaining their right to be forgotten

Data controllers and processors are now required to demonstrate the implementation of technical and organizational measures they have taken in order to ensure data security [127]. The GDPR attempts to increase self-responsibility, and therefore accountability, with a particular reference to the processing of sensitive data, such as health and medical data [127]. Data controllers are required to take measures to ensure that any product or service they provide is fully in line with the key principles of GDPR.

Finally, the data subject is empowered with total control over her data. Consent rules and a number of fundamental rights such as the right to be forgotten have been reinforced or introduced in the GDPR. The data subject's will has become a priority.

Data controllers can outsource data processing activities to data processors. This relationship can be established though contracts or any binding legal act. Personal data should not be made available to third parties without the consent of the data subject. Data processors are not considered as a third-party but rather as an extension of the data controller. In this sense, any activity put in place by the processors with the approval and instructions of the controller is considered to be carried out within the premises of the organization itself [127]. For example, a hospital may act as a data controller of collected personal data of the hospital's patients and outsource an anonymized operation to a trusted data processor without any need to ask anew for the patients' consent.

4.3 Motivations for applications

Blockchain is decentralized, which means that there is no central authority that regulates and governs the recording of the transactions in the ledger. The ledger is maintained by the network participants, in which participation is free, and the global state of the ledger is agreed by a consensus mechanism (§ 3.6). Data on the blockchain are stored in every node of the network. so that each node keeps a copy of the public ledger. Due to data replication there is no single point of failure (SPOF) and the system becomes fault-tolerant and resistant to malicious and denial-of-service attacks (DoS attacks). The users of the network are empowered with total ownership over their assets – ranging from a token, a contract, datasets, medical records, chain-of-evidence documents, or citizenship documents – through a system that guarantees security even in the presence of malicious users.

Blockchain is immutable, meaning that all transactions are irreversible and they can not be altered or deleted once they are confirmed and recorded on the blockchain. The realized transactions are stored in chronological order specifying if a transaction A precedes a

transaction B. That allows the creation of a distributed digital timestamping service. Timestamping is useful for example in financial contracts, establishing precedence for copyright and auction bids [85].

The combination of a timestamping service and the immutability property of the blockchain, along with its consensus mechanism, can lead to the creation of various important properties such as accountability [35], auditability and non-repudiation [122] – the ability to definitively verify authenticity of statements recorded in the blockchain [85]. These properties make the blockchain a distributed immutable, tamper-proof, transparent and audit log that any attempt to tamper with it can be immediately evident and easily detectable. And the users are allowed to confirm and prove whether the service operates in the intended way or not.

Blockchain technology can be used to support a wide variety of applications including document provenance tracking, digital assets, financial services, copyright, voting, distributed storage, donation, education, medical records, data exchange, and Internet of Things (IoT). Distributed consent for research trials can increase anonymous data samples [164] allowing Big Data Analytics in compliance with the GDPR. In addition, blockchain can significantly reduce data transaction costs between entities and increase transaction efficiency. Blockchain technology can be seen as a data sharing and processing system where privacy is mandatory.

Like all technology, blockchain has limitations. It is argued that blockchain is not suited for high performance transactions or as a database replacement. Due to the decentralized nature of the blockchain and its necessity for a consensus mechanism the transaction rate remains quite low compared to financial services [158, 179]. Blockchain is not made for big data. The amount of data that a blockchain can store and process is very limited so off-chain data frameworks are needed.

Private and sensitive data cannot be transmitted outside the premises of the organization responsible for them. Computations over the data must be made by the organization itself. Yet, organizations must be trusted to assure computation is done properly. Outputs of computations can be false either by a malfunction or intentionally. Zero knowledge verifiable computation schemes (§ 2.8.3) can generate proofs of correct computation without revealing private inputs. The proofs can be verified efficiently and with almost no extra overhead [138]. The ability to construct publicly verifiable proofs without exposing sensitive information is crucial to a data processing system which guarantees the privacy of the datasets.

In this thesis we try to address the privacy concerns stemming from data exploitation and data regulation described above. We attempt to utilize the blockchain and a Zero Knowledge verifiable computation scheme and adapt it to our needs in order to provide a solution which can enable data sharing and the data processing in a privacy preserving manner, making anyone accountable for their actions and ensuring proper computation.

5. SOLUTION

In our solution we use a protocol where *datasets* can be *registered* to the system and be made available for *processing*. Anyone can request a data processing over a registered dataset and a processing algorithm of their choice. A request is fulfilled by a *data processor* to whom the dataset is transferred by the *data controller*.

A public blockchain is used as the *controller* of the application. It is responsible for keeping an audit, immutable, tamper-proof and transparent log of all actions of the participants. All participants are *accountable* for their actions and cannot challenge the validity of them. Dataset are stored *off-chain* and data controllers, the owners of datasets, are responsible for their quality, availability and security. To increase trust to data processors, a *zero knowledge verifiable scheme* is used. The data processors are obligated to produce a *proof of correctness of computation*, along with the output of the processing, that the *requestor* in turn verifies.

It has to be noted that the blockchain does not provides an extra security layer concerning datasets. The datasets are stored off-chain, data controllers are responsible for the security of their system and the participants can still collaborate outside the blockchain network. Nevertheless, the use of the blockchain can guarantee accountability, auditability and provenance tracking of the datasets increasing the trust to the system.

5.1 Participants

There are three main roles consisting the application: *the data controller*, the *data processor* and the *data requester*. The first two are also defined in the context of GDPR (§ 4.2). GDPR defines another role, that of data subject; the owner of the data. In our scheme we assume that the data controller already has consent to access or forward the data or she is at the same time the data subject and the data controller.

5.1.1 Data Controller

The data controller is in charge of keeping and managing a data set. It runs on behalf of a *data subject* (person) that authorizes the data controller to access its personal data, with the possibility of forwarding them to a data processor that will be responsible for processing the data on behalf of controller [131].

5.1.2 Data Processor

The data processor is responsible for processing datasets on behalf of the data controller. It listens for data processing requests and returns, along with the output of the process, a Zero Knowledge Proof of correct computation over the requested data set without revealing the dataset itself.

5.1.3 Data Requester

The requester can be any entity that requests a computation over a dataset with the use of a specific algorithm. It can be a research center, a university, a machine learning algorithm or any individual. The requester expects, along with the output, a proof of correct computation over the requested dataset that verifies at the end of the processing.

5.2 Threat model

Before we explain in detail our architecture, we would like to introduce our *thread model* and what exactly our goals are. It is important to understand the possible roles of adversaries, their strengths and their resources.

In our model we assume a public blockchain where the involved entities that interact with a dataset – the data controller and the data processor – are identified and verified through a *public key infrastructure* (PKI) [3]. Furthermore, each of the authenticated entities has certain *trust properties*. The data controller is trusted for integrity and confidentiality and the data processor only for confidentiality.

Adversaries can be divided in 4 categories:

- Malicious data controller
- Malicious data processor
- Malicious requester
- Malicious public user

Each of these entities have different resources and goals. These are explained below.

5.2.1 Malicious data controller

A malicious data controller is a controller who tries to manipulate the results of the processing by either crafting fabricated datasets or working in collusion with the data processor. For example, an adversary could manipulate the dataset to make a classifier produce false negatives [52] or to alter the image representations in a deep neural network (DNN) to mimic those of other natural images [150].

In our solution we do not address those issues. We assume that a data controller is honest and is authorized by a PKI that ensures the quality of the datasets.

5.2.2 Malicious data processor

Similarly to data controllers, data processors may want to manipulate the results of the processing. A malicious data processor could influence the results of the processing with the following means:

- Fake computation

- Process a different dataset
- Use a different algorithm
- Fake results
- Expose a dataset to the public

We aim to fully protect requestors from such malicious actors except of data disclosure.

5.2.3 Malicious requester

The role of a requestor is certain: it can only make requests. Therefore, the only type of attack a malicious requestor can perform is a distributed denial-of-service attack (DDoS) by flooding the network with requests in an attempt to overload data processors and prevent other requests from being fulfilled. DDoS attacks are not addressed in this work and are left for future work (§ 8.3).

5.2.4 Malicious public user

As the blockchain is public, malicious external users should be taken into account. External actors can perform Sybil attacks [60] by impersonating the various actors of the system. All the possible attacks that can be performed by the previous malicious actor they can also be performed by an external user. The PKI assumption prevents a malicious public user to forge data controller and data processor identities.

5.3 Blockchain

A public blockchain is used to keep track the actions of each participant of the system. Each action of the participants is logged in the blockchain, a log that is immutable and irreversible. No one can alter or modify the records and every action can be auditable by anyone. This way, each entity is accountable for each of their action which they cannot later denied it. The blockchain can also serve as a public bulletin board where each action is ordered by time of occurrence and provide dataset timestamping in a decentralized manner. The hash of the dataset can be stored in the blockchain, which serve as a secure proof of the creation and modification time of the dataset.

5.4 Algorithms

The system should support only a set of open-source algorithms that have been analyzed and constructed to be *privacy-preserving* with the use of techniques such as *k-anonymity* [151] and *l-diversity* [4]. This algorithms should return only de-identified aggregated results.

For the moment the supported algorithms by the system are:

1. Sum

2. Average
3. Count
4. Maximum
5. Median
6. Minimum

The above algorithms are not considered to be privacy-preserving as it is outside the scope of the initial prototype and should be considered on future work (§ 8.4).

5.5 Zero-Knowledge Verifiable Computation

Every data processor provides a proof of correctness of execution of a computation on a given dataset without *revealing* the dataset itself. As the processor is obligated to provide a proof – a rational verifier rejects a computation without it – it is impossible for the processor a) to pretend that it done a processing without actually make any computation at all; b) to process a different dataset of its choice; c) to execute a different algorithm other than the requested one; d) to return another result other than the one that returned by the computation. It is evident that a Zero-Knowledge proof of computation plays a crucial role as it address various threats by malicious processors and enforce them to be honest.

Constructions of zkSNARKs require a *one-time trusted setup* in which a common-reference string (CRS) is generated; the public parameters of the system. The CRS is used to construct and verify proofs. Proof generation and verification requires two publicly available keys which derived from the CRS: the evaluation key ek_f and the verification key vk_f. The CRS is different for each function F for which the prover wants to produce a proof of computation. For that reason, a key pair (ek_f, vk_f) needs to be generated one for each of the available algorithms that the processors use.

Anyone who obtains the trapdoor information corresponding to the CRS can produce fake proofs. For that reason, whomever runs the setup should be a trusted entity. Various alternatives to bypass the trusted party – such as secure *multi-party computation* for CRS generation [33] or multi-string models [84] where a set of untrusted authorities generate a random trusted string – have been proposed. In our solution, we assume that the trusted entity, that registers and verifies the parties through the PKI, is also responsible for key generation and distribution. The distribution can be done either by saving the public parameters on the blockchain or in a publicly available server.

In our scheme, the processor executes an algorithm F with public input u and private input w. With the use of the evaluation key ek_f it generates a zero knowledge proof. Then, the requestor with the use of the verification key vk_f can verify that the processor executed correctly the algorithm F on inputs u and w.

First, the data processor want to prove to the verifier that the computation is indeed done on the requested dataset without disclosing the dataset. The dataset is the private input w. For better understanding of the construction of the proof we define a game where tree participants are involved: a trusted oracle, a computationally bounded processor and requestor. The trusted oracle has a list of datasets and the corresponding digests (hashes). The oracle cannot cheat or collaborate with any of the participant and for the same dataset

it produces the same digest. At any moment the oracle selects randomly a dataset, gives it to the processor and announce the hash of the dataset to the requestor. The requestor wants to know if the processor holds the dataset that produce the same digest as the one that was given to it by the oracle. To do that, the requestor ask the processor to compute the hash of the dataset. The computation of the digest of the dataset serves as a proof that shows that the data processor holds the corresponding dataset. The requestor accepts the proof if and only if the hash of the processor match the hash of the oracle and rejects otherwise. The processor is forced to correctly compute the hash of the dataset as it is the only way to convince the requestor – hash functions are one-way and the oracle is trusted.

Let's modify our game and assume now that the oracle gives the hash of the dataset to both the processor and requestor. The processor can easily convince the requestor without computing the hash; it knows the hash beforehand. To countermeasure that, the requestor demands from the processor to produce a proof of verifiable computation. It gives as function F the same hash function H the oracle used to produce the digests of the datasets. Then, it asks the processor to generate a proof for $F(h, d) = H(d)$ where d is the dataset given by the oracle and h the hash of it. The hash is the public input u and the dataset d the private input w. As the processor wants to disclose the dataset d it produces a zero knowledge proof. The requestor verifies the proof and accept if and only if the proof is valid. Again, the processor can not do anything else but to comply and compute the digest.

Ideally, we would like the trusted oracle to be the data controller and make an indistinguishability assumption between them. That is not the case. The data controller is trusted to publish the correct digest of the dataset in the blockchain and the immutability property of the blockchain prevent a malicious processor to change the digest to its preferences.

What remains, is to include inside F the processing procedure of the dataset. The algorithm F with private inputs d and public input h, for which a zero knowledge proof of correct computation is generated, consists of a) hash generation of dataset d; b) dataset processing.

Let d be a private dataset, H a cryptographic hash function and $h = H(d)$ the digest of d over H. Let $\mathcal{P}$ be the prover (data processor) and $\mathcal{V}$ the verifier (data requestor). Prover $\mathcal{P}$ produces a zkSNARK proof (§ 2.8.4) π for the following *NP statement*:

Given the public digest h, an outsourced function F and an output y I know a private dataset d such that:

1. $H(d) = h$
2. $F(d) = y$

The irreversibility of cryptographic hash function in conjunction with the immutability of the blockchain and the trust for data integrity to the data controller, that produced the digest of the dataset in the first place, guarantees that indeed the prover $\mathcal{P}$ processed the requested dataset without revealing it.

The proof π is publicly available for anyone to verify. In our solution, the result of the computation remains private and is encrypted with the public key of the requestor. Thus, only the requestor can verify the proof – the raw output is needed by the verification algorithm. A malicious adversary who wants to learn the output y can potentially do this by brute-force attacking y using the public parameters of the zkSNARK setting. A realistic example

could be when F computes the summation of a list of integers and return 0 if the sum is even and 1 if it is odd. An adversary can use the verification algorithm for both possible outputs and guess y correctly. To avoid this, a random *salt* $s \xleftarrow{r} \{0,1\}^l$, with security parameter l, is used as an additional input of the proof. Each proof should use a unique salt. Although, the processor generates the salt before constructing a proof and is private to anybody than the requestor, in the zkSNARK setting it is considered as a public input.

The proof generation procedure is summarized in Algorithm 1.

Algorithm 1 Zero Knowledge Proof

function compute(F,d, h, ek_f, s)
    $h^{'} \leftarrow H(d)$ ▷ Compute the digest of the dataset
    $y \leftarrow F(d)$ ▷ Process dataset
    out $\leftarrow (y, h^{'}, s)$
    return (π, out) ▷ Return results
end function
procedure zkp(ek_f, F, u, w)
    $d \leftarrow w$ ▷ Private input
    $(h, s) \leftarrow u$ ▷ Public input
    $(\pi, \text{out}) \leftarrow$ compute(F, d, h, ek_f, s)
end procedure

5.6 Dataset Registration

Naturally, for datasets to be available for processing a data registration process is needed. Anyone who register a dataset automatically becomes a data controller who is responsible for the availability and quality of it. The dataset should be publicly available on any location of their choice provided that the controller expose an API for data retrieval. The location can be in a distributed file system [22], a decentralized cloud storage [173] or a central server. As long as the involved participants communicate over the same protocol the choice is irrelevant; the system is agnostic concerning data storage.

The confidentiality of the dataset must rely solely on cryptographic primitives with strong guarantees. Therefore, every dataset, before stored, is encrypted using the symmetric encryption algorithm `AES` with `256-bit` key length and `CTR` as mode of operation. For each dataset a different encryption key is created and used. This approach bear the burden of key generation and management but on the other hand if an attacker manages to obtain a key the security of other datasets is not compromised [154].The hash of contents of the unencrypted dataset is computed, and stored in the blockchain, using the cryptographic hash function `SHA256`. As discussed in 5.5 the hash of the dataset plays an important role in the application.

A dataset is actually registered and available when a *register* transaction is sent to the blockchain signed by the data controller. The transaction needs to include the name of the *dataset*, a *category*, the *location* (URI) and the *digest* (hash) of the file. The hash of the dataset is used as a *unique persistent identifier* to which all participants refer to.

The data registration procedure is summarized in Algorithm 2.

Algorithm 2 Dataset registration

```
1:  function save(file)
2:      h ← H(file)                                  ▷ Compute the digest of the dataset
3:      k ← G                                        ▷ Generate symmetric key
4:      c ← Enc_k(file)                              ▷ Encrypt dataset
5:      uri ← store(c)                               ▷ Store dataset
6:      return (h, k, uri)
7:  end function
8:  function broadcast(name, uri, category, hash)
9:      h_meta ← H(name || uri || category || hash)  ▷ Compute the hash of the metadata
10:     tx ← (name, uri, category, hash, h_meta)     ▷ Create transaction
11:     tx.send()                                    ▷ Send transaction
12:     return tx
13: end function
14: procedure register(name, file, category)
15:     (h, k, uri) ← save(file)                     ▷ Save dataset
16:     tx ← broadcast(name, uri, category, h)       ▷ Register dataset
17:     return (tx, k)
18: end procedure
```

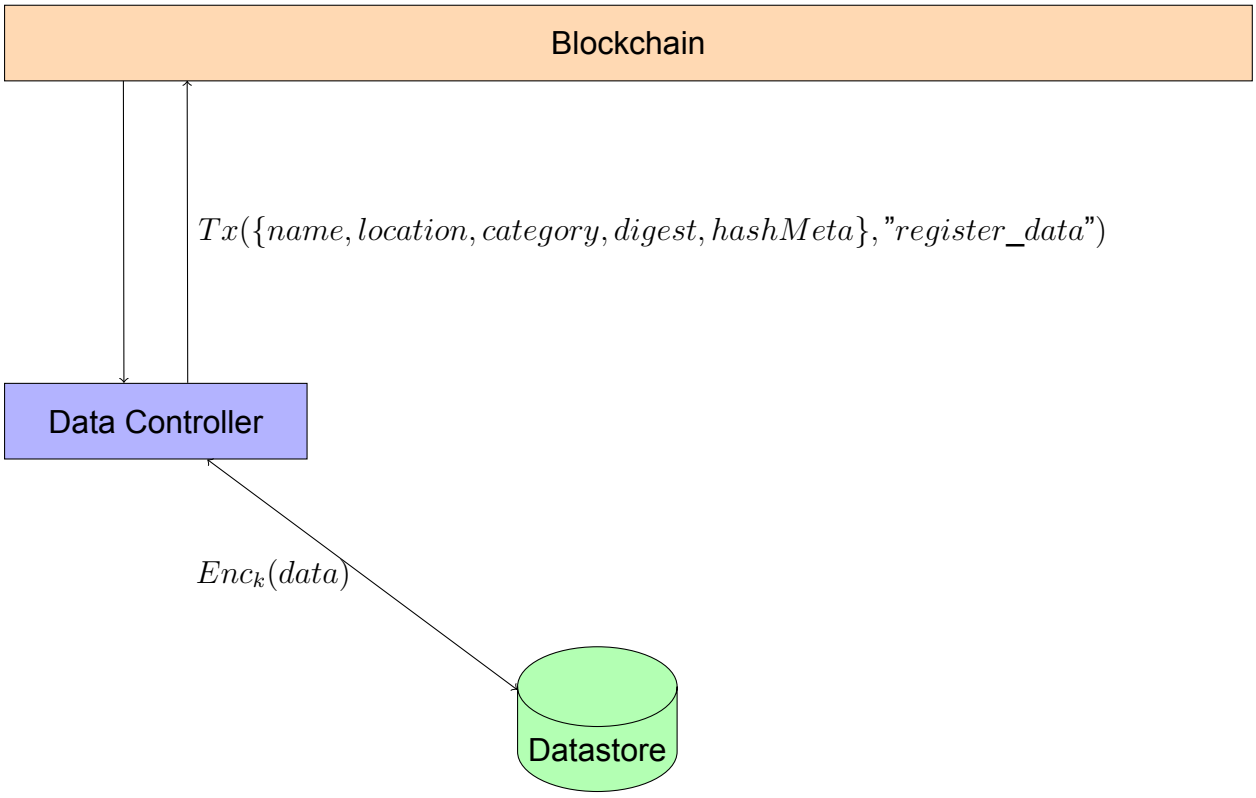


Figure 33: Data registration

5.7 Entity Registration

The entity registration procedure can be done only by the trusted entity that deploys the smart contracts on the blockchain. The smart contracts are implemented in a way that only the creator of them can call the registration function. The trusted entity can register a data controller or a data processor by signing a transaction that call the registration function

with arguments the entity's name and public key. Only registered entities can register or process a dataset.

Algorithm 3 Entity registration

1: **procedure** register_entity(name, p_k)
2: $tx \leftarrow$ (name, p_k)
3: tx.send()
4: **return** tx
5: **end procedure**

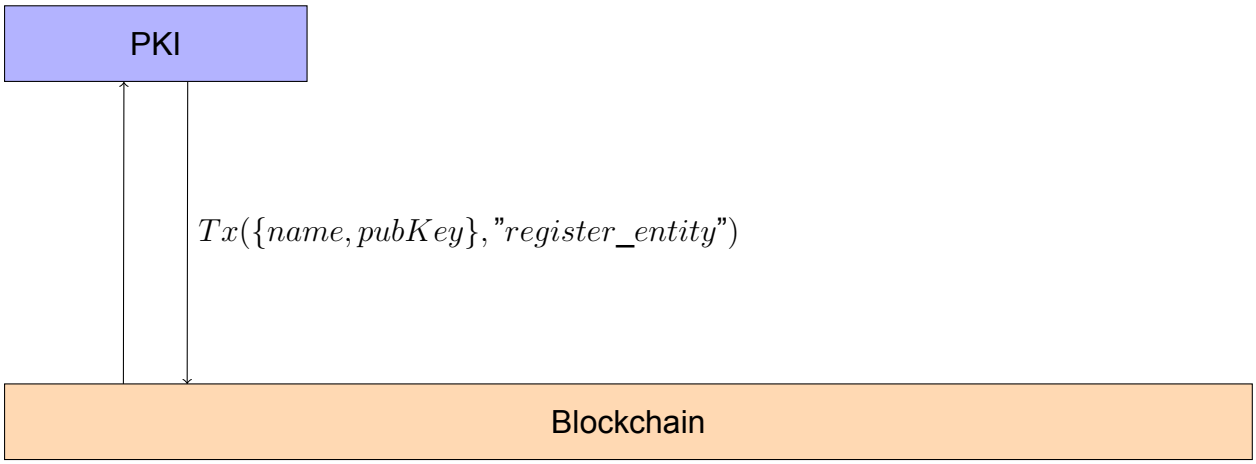


Figure 34: Entity Registration

5.8 Request for processing

A *request for processing* is the procedure where a participant of the network request a specific processing algorithm to be performed on a dataset of its choice. The requestor can only choose datasets that are registered on the network and publicly available on the blockchain. A request for data processing is registered when the requestor signs and sent a request transaction to the network. In its payload the public key of the requestor must be included. The public key is needed by the processor to be able to encrypt data processing results as they must remain private from other parties. When a request is registered, a `request` event is emitted notifying the interested parties. All data controllers are listening to `request` events and are responsible for notifying a data processor of their choice that in turn will fulfill the request. The data controller, responsible of the requested dataset, selects *randomly* a data processor and encrypts the symmetric key of the dataset with the public key of the data processor. The choice of the data processor could also be done *sequentially*; the data controller choose the next in line data processor; selection by popularity in the existence of a decentralized ranking system is a another viable option. A transaction is sent to the blockchain to notify the selected data processor passing along the id of the request and the encrypted symmetric key.

The request for processing procedure is summarized in Algorithm 4.

Algorithm 4 Request for processing

```
1: function notify(e)
2:     p ←r {p_1, p_2, ... p_n}                    ▷ Select randomly processor
3:     (requestID) ← e
4:     tx ← (p)
5:     (pk_p) ← tx.send()                          ▷ Get processor public key
6:     c ← Enc_{pk_p}(k)                           ▷ Encrypt symmetric key
7:     tx ← (requestID, c)                         ▷ Notify processor
8:     tx.send()
9:     return tx
10: end function
11: procedure watch()
12:     while e ∈ events['request'] do              ▷ Listen data processing requests
13:         ▷ Check if the controller is the owner of the dataset
14:         if isOwner(e.datasetOwner) then
15:             notify(e)                           ▷ Notify processor
16:         end if
17:     end while
18: end procedure
19: procedure request(datasetID, algorithmID, p_k)
20:     tx ← (datasetID, algorithmID, p_k)          ▷ Request for processing
21:     tx.send()
22:     return tx
23: end procedure
```

5.9 Dataset processing

As it has already been mentioned, the role of a data processor is crucial. They decrease the pressure over data controllers which they exploit them by entrusting specific data processing operations on behalf of data requestors. Data processors can only perform pre-agreed algorithms (§ 5.4) on registered datasets. A processor is constantly listening for `process` events related to it. As seen in § 5.8, the controller *notify* the processor by emitting those events including the ID of the request and the encrypted symmetric key of the dataset. The processor, having all the needed information, signs a transaction to get the details of that request; the *dataset ID* and the *algorithm ID*. Another transaction is made to get the location and the hash of the dataset. As all informations are saved only in the blockchain this transactions are needed. The processor is ready to process the dataset. At first, it decrypts the symmetric key with its private key sk_p, it downloads the dataset from the provided location and decrypts it. Assuming no errors, the computation is started and a proof, as analyzed in 5.5, is generated with the evaluation key ek_f. The results are encrypted with the public key of the requestor pk_r. Final, a transaction is sent to the blockchain with the proof π and the encrypted results and an event is emitted to notify the requestor for the completion of the data processing.

The data processing procedure is summarized in Algorithm 5.

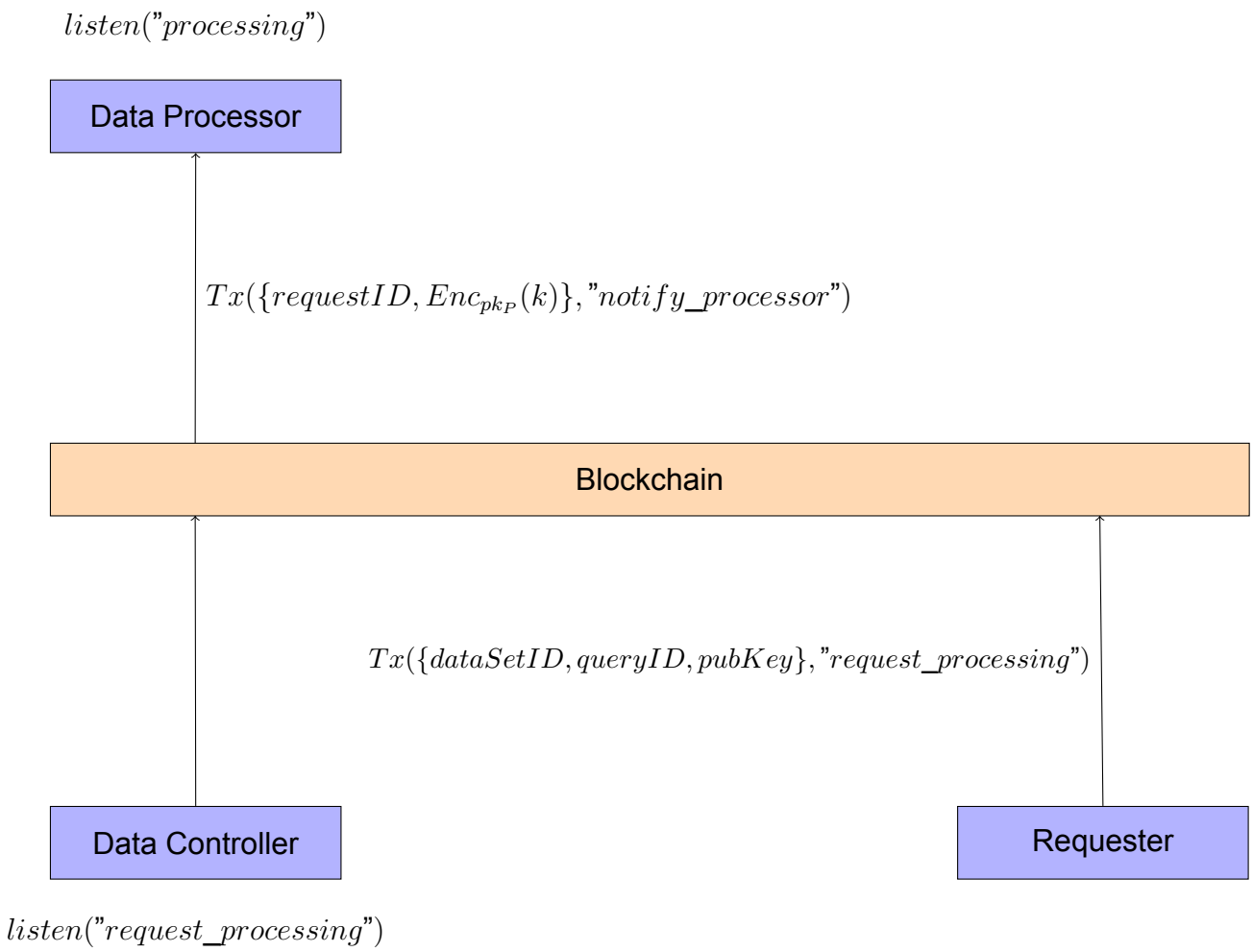


Figure 35: Request for processing

5.10 Proof verification

As zkSNARKs are non-interactive, the requestor can verify at any time the correctness of the processing. It can either watch for `process_done` events or by manually checking in the blockchain if the processing request has finished. Either way, is in its best interest to verify the proof. Providing the request ID the requestor can get the proof and the encrypted results which in turn decrypts and verifies the proof with the verification key vk_f. In case of verification failure the processing results should be rejected.

The verification procedure is summarized in Algorithm 6.

Algorithm 5 Dataset processing

procedure process(e)
  $(\text{requestID}, \text{algorithmID}, c_k, pk_r) \leftarrow e$
  $tx \leftarrow (\text{requestID})$ ▷ Get dataset id
  $(\text{datasetID}, \text{algorithmID}) \leftarrow (tx)$ ▷ Get dataset info
  $(\text{location}, h) \leftarrow (tx)$
  $k \leftarrow Dec_{sk_p}(c_k)$ ▷ Decrypt symmetric key
  $c_d \leftarrow$ get(location) ▷ Get encrypted dataset
  $d \leftarrow Dec_k(c_d)$ ▷ Decrypt dataset
  ▷ Process dataset and get proof of computation
  $s \xleftarrow{r} \{0,1\}^l$ ▷ Salt generation with security parameter
  $F \leftarrow$ algorithms['algorithmID']
  $u \leftarrow (h, s)$ ▷ Public input
  $w \leftarrow d$ ▷ Private input
  $(\pi, \text{out}) \leftarrow zkp(ek_f, F, u, w)$ ▷ Algorithm 1
  $tx \leftarrow (\pi, Enc_{pk_r}(\text{out}))$ ▷ Send results to requestor
  tx.send()
end procedure
procedure watch()
  while $e \in$ events['process'] **do** ▷ Listen data processing notifications
    process(e) ▷ Process request
  end while
end procedure

Algorithm 6 Proof verification

procedure verify(e)
  $(\text{requestID}) \leftarrow e$
  $(\pi, c, \text{datasetID}) \leftarrow tx(\text{requestID}).send()$ ▷ Get request info
  $h \leftarrow tx(\text{datasetID}).send()$ ▷ Get dataset info
  $(\pi, (y, h^{'}, s)) \leftarrow Dec_{sk_r}(c)$ ▷ Decrypt results
  if $h^{'}$!== h **then** ▷ Reject if digest not match
    reject
  end if
  $valid \leftarrow \_verify(\pi, (y, h^{'}), (h, s), vk_f)$ ▷ Verify proof
  if !valid **then** ▷ Reject if not valid
    reject
  end if
end procedure
procedure watch()
  while $e \in$ events['process_done'] **do** ▷ Listen data processing completion
    verify(e) ▷ Process request
  end while
end procedure

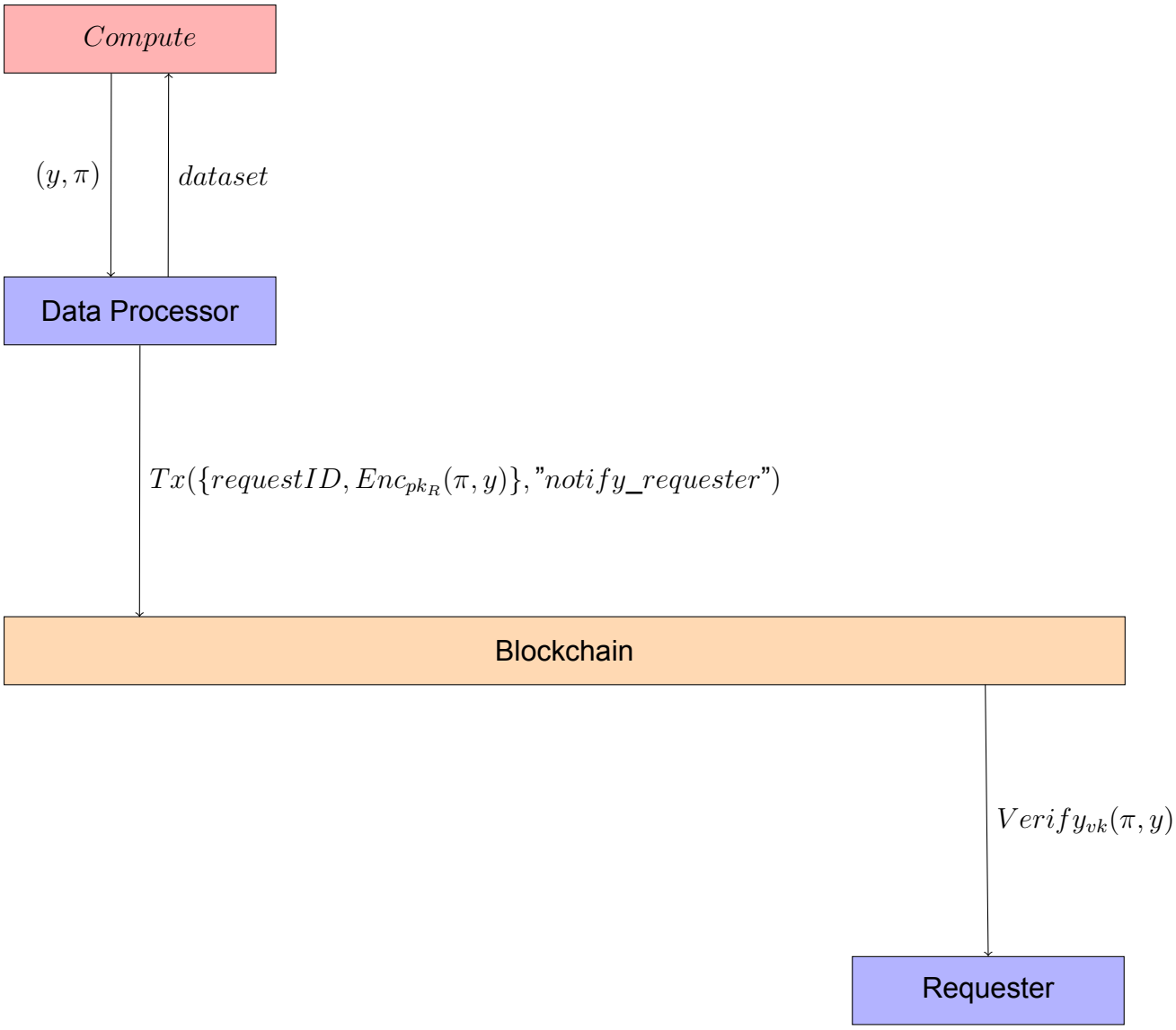


Figure 36: Data processing

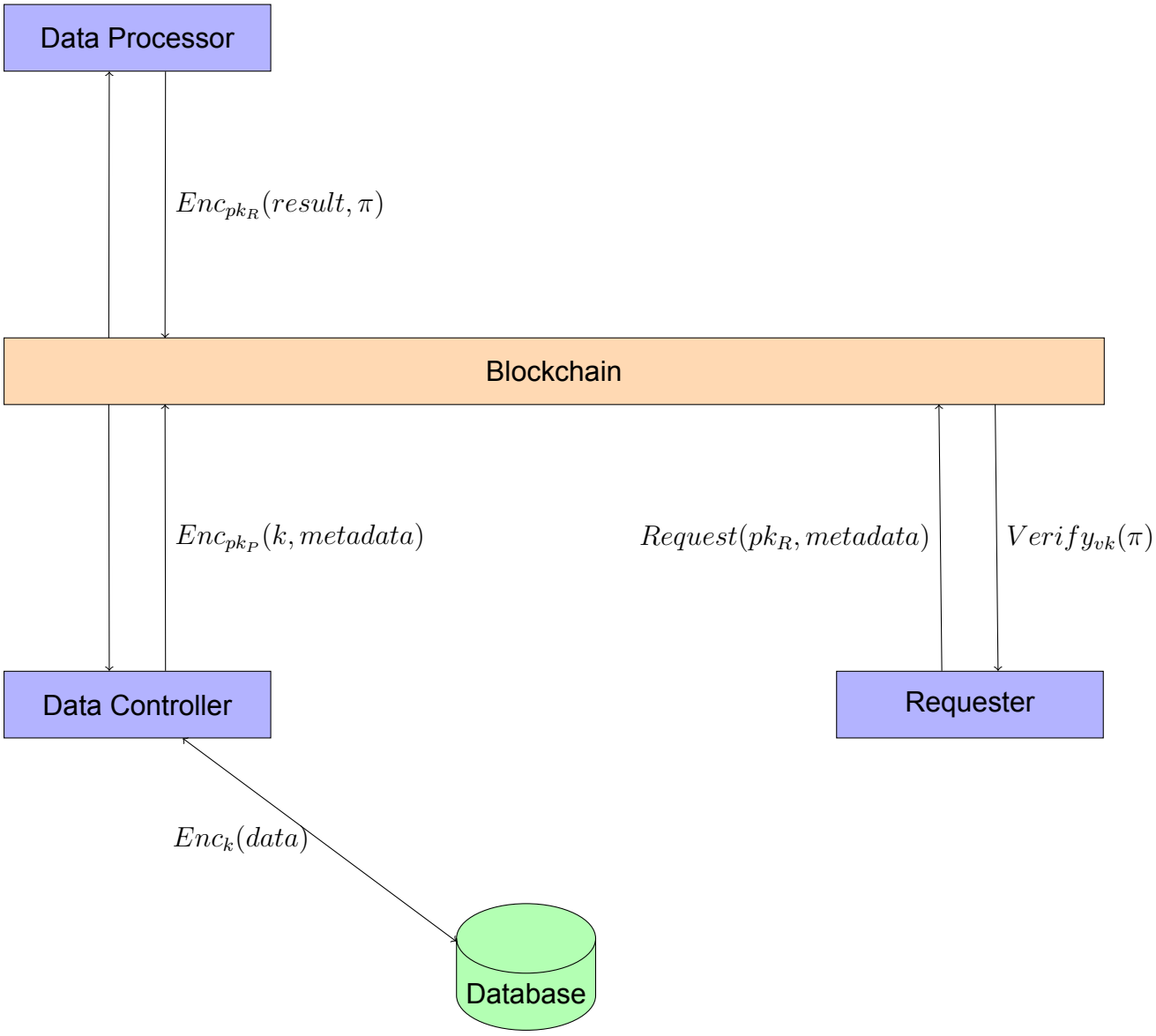


Figure 37: Architecture

6. IMPLEMENTATION

To cover the needs of the data sharing and processing ecosystem the application is separated in two parts: the *off-chain network* and the *blockchain network*. The off-chain network is responsible for dataset storage, dataset distribution, and dataset processing. The blockchain is responsible for recording dataset registrations, requests for data processing and data processing outputs along with their proofs of computations. For that reason, every node of the network – *a data share node* – is connected to the blockchain and track every transaction related to the application. The node can track and index every registered dataset, all requests for data processing and all processing outputs and proofs. A node, depending on its role, acts accordingly when a blockchain transaction is broadcasted in the network.

A variety of tools and APIs are made to implement the protocol as described in Chapter 5. They are analyzed in the following sections.

6.1 API

A server is implemented to aid developers and users and is responsible to: a) expose a RESTful API that facilitates the interaction with the application, and b) listen and record to a relational database all the events of the application emitted by the blockchain.

The existence of such server is not a threat to the decentralization of the system. The application is governed by the smart contracts deployed on the blockchain. Anyone can interact directly with the blockchain or implement their own server or client.

6.1.1 RESTful API

A RESTful API is provided to facilitate communication between any application, that follows the REST architecture, and the data sharing application. The REST API expose the blockchain business network that can be easily consumed by HTTP or REST clients. That way, any developer familiar with existing web technologies and frameworks is not obligated to learn the interval mechanisms of the blockchain system to develop and deploy applications atop.

Through the REST API one can:

- Get all datasets
- Register a dataset (as pending)
- Get a specific dataset
- Get all processing requests
- Register a request for processing (as pending)
- Get a specific request
- Get all processors

- Register a processor (as pending)
- Get all controllers
- Register a controller (as pending)
- Get all blockchain accounts

The REST API expose a set of routes each one having an HTTP method and a URI. All available routes are shown at Table 6.

Table 6: RESTful API Routes

Method	URI
GET	/
GET	/contracts
GET	/datastore
GET	/datastore/{data}
POST	/datastore
GET	/accounts
GET	/accounts/{account}
GET	/requests
GET	/requests/{request}
POST	/requests
GET	/processors
POST	/processors
GET	/controllers
POST	/controllers

All blockchain transaction made by a user must be sign with her signing key. As keys should be private and be managed only by the user itself, the server exposing the REST API cannot make a transaction on behalf of the user. For that reason, all `POST` actions create a database record where the status of the action is set as `pending`.

6.1.2 Database

The server is registered to all available events (§ 6.7.7) of the system that are emitted by the blockchain. When an event is triggered a record of the action is saved to a relational database. The database's scheme is shown in Figure 38.

6.2 Distributed Application

The distributed application is a web-based application providing a graphical user interface (UI) to facilitate the data sharing platform usage. The goal of the distributed application is to provide to the end-user an easy, self-explanatory and efficient way of interacting with the data sharing ecosystem. As most users are already familiar with the use of web applications and platforms, such as webmails, cloud services and social media, such interfaces makes the user feel accustomed. The only requirement is to have an ethereum account and installed the metamask wallet [126].

The distributed application consumes the REST API and yields the same functionalities.

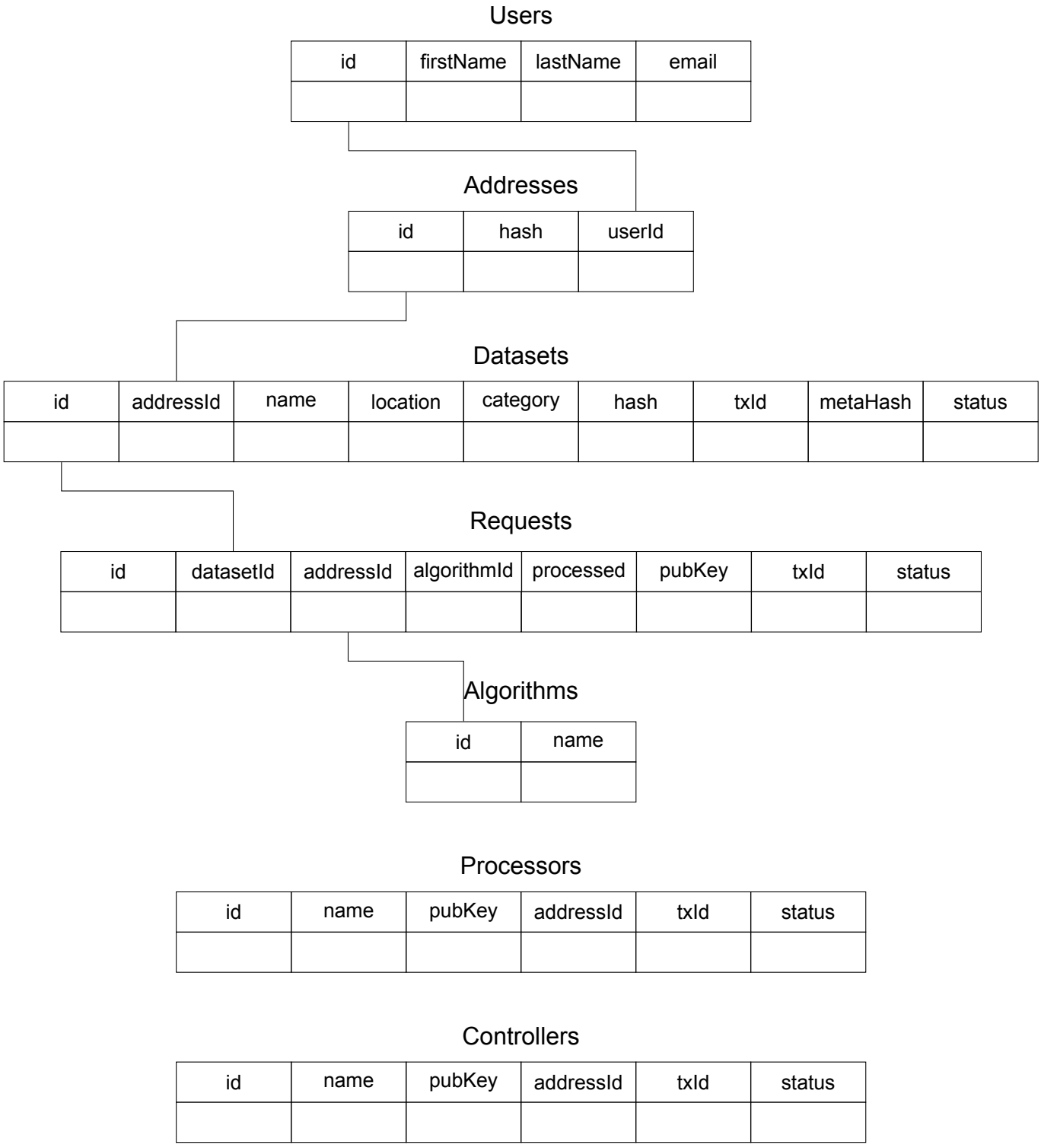


Figure 38: Database scheme

6.3 Controller

The controller is a data sharing node that is responsible for providing a dataset for data processing. It listens for processing requests and forwards datasets to data processors. Is implemented as described in § 5.1.1 and § 5.8.

6.4 Processor

The processor is a data sharing node that listens to data processing notifications pushed by a data controller. It processes datasets and produces Zero Knowledge proofs of correctness of computation. Is implemented as described in § 5.1.2 and § 5.9.

6.5 Libraries

Various libraries have been build to facilitate the interaction with the various components of the system. Each one is analyzed in the following sections.

6.5.1 Blockchain

The blockchain library offers a set of functionalities that makes the communication with any blockchain implementation easier. Is a wrapper library over blockchain specific libraries, such as Ethereum's `web3`, and it's main purpose is to be blockchain agnostic allowing the use of different Blockchains. For the moment only the Ethereum blockchain is supported.

Abstract Functions:

- `isConnected`: Return true if a connection to a blockchain node exists and false otherwise.
- `getProvider`: Get the current provider (blockchain node)
- `setProvider`: Set provider (blockchain node)
- `getBalance`: Get the balance of an address
- `setDefaultAccount`: Set the default address
- `getDefaultAccount`: Get the default address
- `getAccounts`: Get all accounts
- `getLibInstance`: Get the instance of the blockchain library
- `getFilter`: Get a filter object
- `toBytes`: Covert a string to bytes
- `fromBytes`: Covert bytes to a string
- `registerDataSet`: Register a dataset
- `registerProcessor`: Register a processor
- `registerController`: Register a controller
- `requestProcessing`: Request a data processing
- `notifyProcessor`: Notify a data processor for data processing

6.5.2 Crypto

The crypto library provides various cryptographic functionalities. It wraps the `SJCL` and the `Node.js Crypto module` libraries.

Functions:

- `generateKeyPair`: Generate an asymmetric encryption key pair
- `generateKey`: Generate a symmetric encryption key
- `pubEncrypt`: Encrypt a string with a public key with the use of ECC
- `pubDecrypt`: Decrypt a ciphertext with a secret key with the use of ECC
- `encryptFile`: Encrypt a file with a symmetric key with the use of AES256
- `decryptFile`: Decrypt a file with a symmetric key with the use of AES256
- `bytesToHex`: Convert bytes to hex
- `hextoBytes`: Convert hex to bytes
- `hashFile`: Hash a file with the use of SHA256
- `hash`: Hash an array of values with the use of SHA256

6.6 Command-line interface

A command-line interface that provides a set of useful commands for the data sharing application.

Usage:

```
$ node data-cli <options>
```

Options:

- `-v` or `--version`: Output the version number
- `-g` or `--generate-keys`: Generate an asymmetric
- `-k` or `--generate-key`: Generate a symmetric key
- `-d` or `--dummy-file <file>`: Generate a 45MB dummy file
- `-e` or `--encrypt-file <file>`: Encrypt a file
- `-a` or `--evaluation <bytes>`: Evaluate gas cost of bytes on Ethereum
- `-s` or `--hash <hash>`: Hash an array of values with SHA256
- `-f` or `--hash-file <file>`: Hash a file with SHA256
- `-h` or `--help`: Output usage information

6.7 Smart contracts

The data sharing application is governed by a smart contract deployed on the Ethereum network. The smart contract contains the logic of the application. Thought the smart contract one can register a new dataset, request for data processing, register a data controller or a data processor, and save a ZKP proof along with the output of the computation. The data sharing smart contract is publicly available and can be used by any other contract in the network.

6.7.1 Data set registration

A data controller registers a new dataset to the application by calling the `registerDataSet` function with the following arguments:

- `hash`: The SHA256 of the raw data set. It serves as a unique identifier of the dataset: the `dataSetID`.
- `name`: The name of the dataset. It can be an arbitrary string
- `location`: The URI of the location of the data set
- `category`: The category of the data set. It can be an arbitrary string
- `metaHash`: The SHA256 of the submitted values: `hash`, `name`, `location` and `category`

```
function registerDataSet(
  bytes32 hash,
  bytes32 name,
  string location,
  bytes32 category,
  bytes32 metaHash
) public returns (bool success);
```

Code 4: Data set registration function

The address of the controller is taken from the `msg.sender` field.

6.7.2 Request for processing

A request for data processing consists of two parts: Requesting for data processing and notifying the data processor of that request.

A data processing request is made by calling the `requestProcessing` function with the following arguments:

- `_dataSetID`: The id of the dataset of interest
- `algorithmID`: The id of the algorithm that the processor will apply over the data set
- `pubKey`: The public key of the requestor

A unique identifier of the request is created by applying the `SHA256` hash function to the concatenation of the dataset's id with the address of the requestor. In particular, as `SHA256(_dataSetID, msg.sender)`. The address of the requestor is taken from the `msg.sender` field.

In order to notify a data processor that a request for data processing is pending, a data controller calls the `notifyProcessor` function with the following arguments:

- `_processorAddress`: The address of the data processor that will process the dataset
- `_requestID`: The id of the request
- `encryptedKey`: The encrypted symmetric key with which the dataset has been encrypted

```
function requestProcessing(
  bytes32 _dataSetID,
  bytes32 algorithmID,
  string pubKey
) public returns (bool success);

function notifyProcessor(
  address _processorAddress,
  bytes32 _requestID,
  string encryptedKey
) public returns (bool success);
```

Code 5: Request for processing functions

6.7.3 Processor Registration

A data processor can be registered by a transaction that invokes the `registerProcessor` function with the following inputs:

- `_processorAddress`: The address of the data processor
- `name`: The name of the data processor. It can be an arbitrary string
- `pubKey`: The public key of the data processor

Only the contract creator can execute the `registerProcessor` function.

```
function registerProcessor(
  address _processorAddress,
  bytes32 name,
  string pubKey
) public returns (bool success);
```

Code 6: Data processor registration function

6.7.4 Data Controller Registration

Likewise, a data processor can be registered by a transaction that invokes the `registerController` function with the following inputs:

- `_controllerAddress`: The address of the data controller
- `name`: The name of the data controller. It can be an arbitrary string
- `pubKey`: The public key of the data controller

Again, only the contract creator can execute the `registerController` function.

```
function registerController(
  address _controllerAddress,
  bytes32 name,
  string pubKey
) public returns (bool success);
```

Code 7: Data controller registration function

6.7.5 General functions

Various general function are provided to facilitate the usage of the system.

- `getDataSetInfo`: Given a data set id it returns all the information of the specific data set
- `getRequestInfo`: Given a request id it returns all the information of the specific request
- `getController`: Given a data controller address it returns all the information of the specific data controller
- `getProcessor`: Given a data processor address it returns all the information of the specific data processor

```
function getDataSetInfo(bytes32 _dataSetID)
  public
  view
  returns(
      bytes32 name,
      string location,
      bytes32 category,
      bytes32 metaHash,
      address controller
    );

function getRequestInfo(bytes32 _requestID)
  public
  view
  returns(
    bytes32 dataSetID,
    address requestor,
    bool hasProof,
```

```
        bool processed,
        bytes32 algorithmID,
        string pubKey
    );

    function getController(address _controller)
      public
      view
      returns(
        bytes32 name,
        string pubKey
    );

    function getProcessor(address _processor)
    public
    view
    returns(
      bytes32 name,
      string pubKey
    );
```

Code 8: General functions

6.7.6 Zero Knowledge Proof

A data processor can save the ZKP proof and the output of the computation by calling the `addProof` function with the following arguments:

- `_requestID`: The request id
- `proof`: The proof in hex format
- `output`: The encrypted output

```
function addProof(
  bytes32 _requestID,
  string proof,
  string output
) public returns (bool success);
```

Code 9: Data sharing application events

6.7.7 Events

Every function that is invoked by a signed transaction emits an event notifying the occurrence of the action by a specific participant. Every node of the network can register and listen to this events.

In particular:

- `NewDataSet`: A data set is registered
- `NewProvider`: A data controller is registered

- `NewProcessor`: A data processor is registered
- `NewRequest`: A request for data processing is made
- `Process`: A notification for data processing is sent to a specific processor

```
event NewDataSet(
    bytes32 hash,
    bytes32 name,
    string location,
    bytes32 category,
    bytes32 metaHash,
    address controller
);

event NewController(
    address _controllerAddress,
    bytes32 name,
    string pubKey
);

event NewProcessor(
    address _processorAddress,
    bytes32 name,
    string pubKey
);

event NewRequest(
    bytes32 _requestID,
    bytes32 _dataSetID,
    address _requestor,
    bytes32 algorithmID,
    string pubKey
);
event Process(
  address _processorAddress,
  bytes32 _requestID,
  string encryptedKey
);
```

Code 10: Data sharing application events

6.8 Zero Knowledge Proofs

To generate and verify zkSNARKs proofs the `Pequin` library [140] is used. `Pequin` is a toolchain to verifiably execute programs expressed in the C programming language. `Pequin` consists of a front-end and a back-end. The front-end takes C programs and transforms them to a set of arithmetic constraints as described in 2.8.4. The back-end in Pequin is a zk-SNARK. Pequin uses `SCIPR Lab's libsnark` [111], which is an optimized implementation of the back-end of `Pinocchio` [138], itself a refinement and implementation of `GGPR` [80].

Each of the supported algorithm (§ 5.4) is written in the C programming language. A trusted setup has been run and a key pair (e_k, v_k) has been generated for each of the available algorithms. The evaluation keys and the compiled programs are available to all

data processors as they are necessary components for proof generation and data computation. The verification keys are also publicly available to all requestors important to proof verification.

A set of various bash scripts are made to facilitate the data processors and the data requestors. They provide simple commands that abstract the interaction with the `Pequin` library.

6.9 Programming details

The data sharing ecosystem (§ 5) required various frameworks, technologies, programming languages and libraries.

In particular:

1. RESTful API: `Node.js v8.9.4` [76], `Express 4.16.2` [75]
2. Command-line interface: `Node.js v8.9.4`
3. Distributed application: `React 15.6.1` [95], `Redux 3.7.2` [145] and `Bootstrap 4.0.0` [12]
4. Processor node: `Node.js v8.9.4`
5. Controller node: `Node.js v8.9.4`
6. Libraries:
   (a) Blockchain: `web3 0.20.4` [67]
   (b) Crypto: `SJCL 1.0.7` [160] and `Node.js v8.9.4 Crypto Module`
7. Smart contracts: `Truffle 4.1.0` [45] and `Solidity 0.4.24` [74]
8. Zero Knowledge Proofs: `Pequin` [140] and `libsnark` [111]
9. Ethereum Blockchain: `Ganache CLI v6.0.3 (ganache-core: 2.0.2)` [44]

All code written in `JavaScript` follows the `ES6 (ECMAScript 2015)` [96] standard and the `StandardJS` [2] style guide.

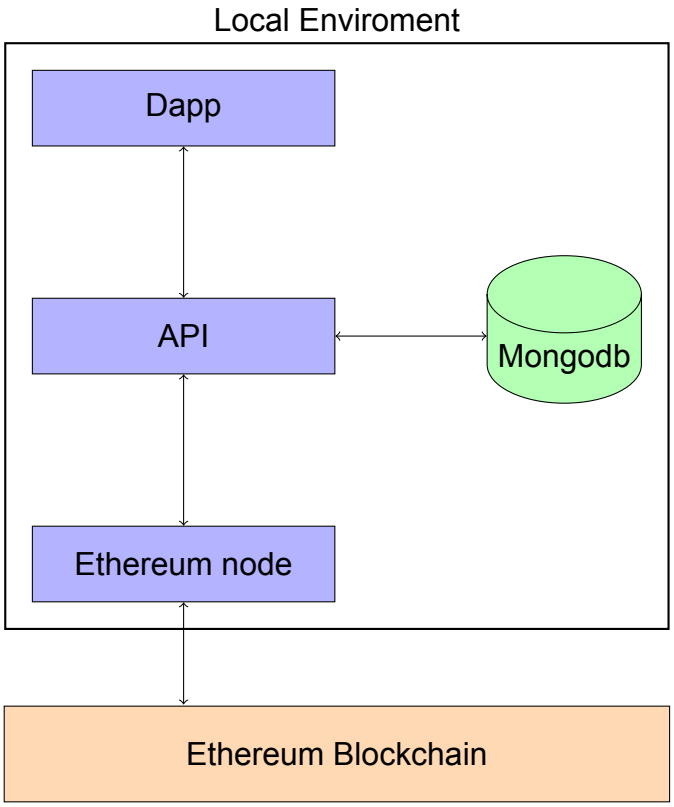


Figure 39: REST API & Dapp implemenation scheme

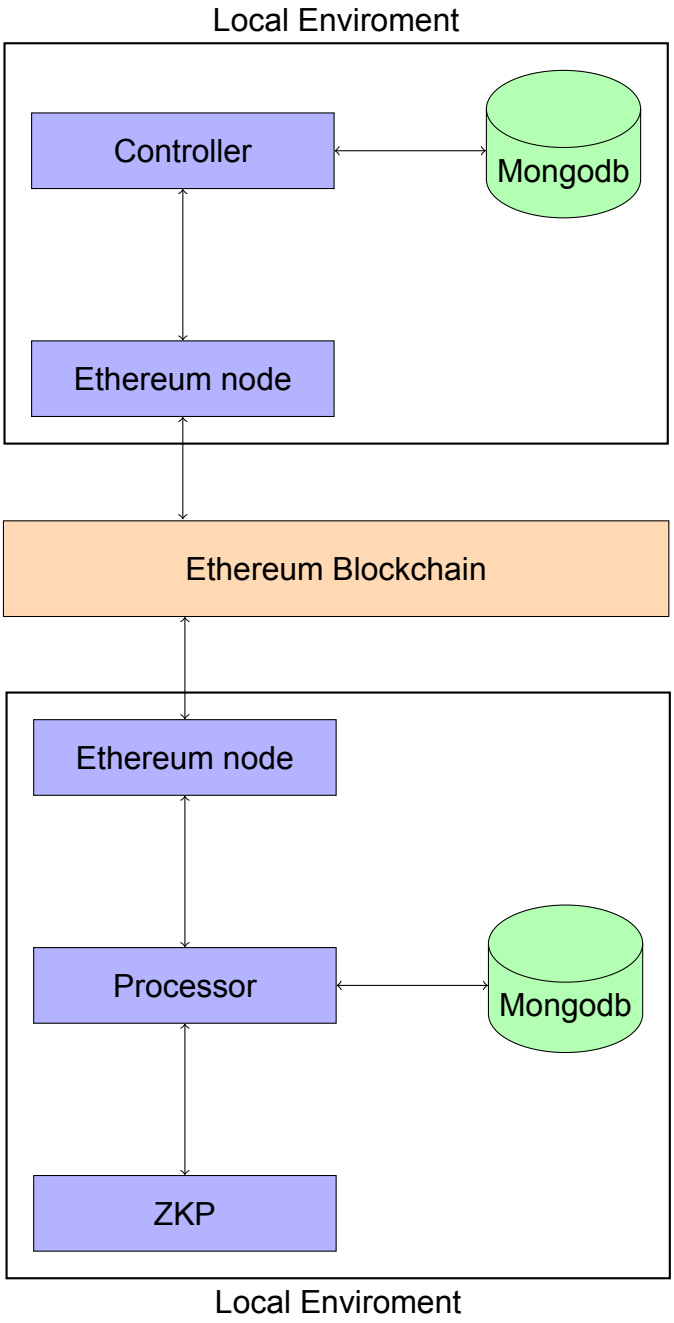


Figure 40: Data processor and data controller implemenation scheme

7. EVALUATION

We measure the efficiency of our solution via various experiments. Storing data on Ethereum, and in blockchain in general, is expensive. For that reason, the first series of experiments are made to measure the cost of saving data in smart contracts by comparing and analyze various methods and alternatives. Second, we measure the cost of deploying the application on Ethereum and the cost of each of the individual operations of it. Lastly, a series of benchmarks are made to measure the time and the size of computational proofs for each of the predefined algorithms supported by the data processors.

All experiments are conducted on MacBook Pro 2016 having a `2GHz Intel Core i5 (2 cores)` CPU with `8GB (1867 MHz LPDDR3)` of memory and a `256GB PCI-Express SSD`. We use `Ganache CLI v6.0.3 (ganache-core: 2.0.2)` as local Ethereum blockchain testnet and all smart contract are compiled with `Solidity 0.4.24`.

7.1 Benchmarks

7.1.1 Data storage

We measure the cost of storing raw data on the blockchain for each one of the following sizes: `32 bytes`, `1KB`, `1MB`, `10MB`, `100MB` and `1GB`. The size of `32 bytes` is included as EVM words are of `256-bits`. Table 7 reports the results.

As size grows in multiples of ten the cost is growing significantly. Storing 1GB of data in Ethereum costs around 13.000.000 US dollars make it impossible and unpractical to store big data in the blockchain. An alternative is to store the data in an off-chain infrastructure (i.e IPFS [22], Storj [173] or Filecoin [112]) and store a reference to the data in the blockchain – a hash pointer. Storing a fixed size reference is much cheaper than saving arbitrary size datasets. Another alternative is to use the logging system of Ethereum whose cost of use is much cheaper than direct storage.

In an attempt to reduce storage costs, various experiments were conducted to compare different storage techniques. Four basic technique have been evaluated: data storage, data hash pointer storage, data logging, data hash pointer logging. Tables 8 to 12 shows the cost of the four possible ways of storing data for sizes of `32 bytes`, `1KB`, and `1MB`. The size of hash value is constant and `32 bytes`. Logging instead of saving the hash value to storage is cheaper by 35.42%. Logging raw data far exceeds direct data storage by an average of 700%.

At the time of the experiments the price of Ether is 942.99 US dollars and the gas price is set at 2 Gwei.

7.1.2 Application costs

Each action of the participants has a cost in gas. Table 13 shows the costs of each action of the application. The most expensive action is dataset registration. It costs 233.487 gas. The reason is that various dataset's metadata are saved upon registration. Nevertheless, is a negligible cost considering the advantages of data sharing in privacy preserving manner.

Table 7: Data storage costs.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei

Size	Gas	USD
32 bytes	20.000	$0.037
1KB	724.664	$1.357
1MB	697.325.562	$1,305.393
10MB	7.000.000.000	$13,104
100MB	70.000.000.000	$131,040
1GB	700.000.000.000	$13,104,000

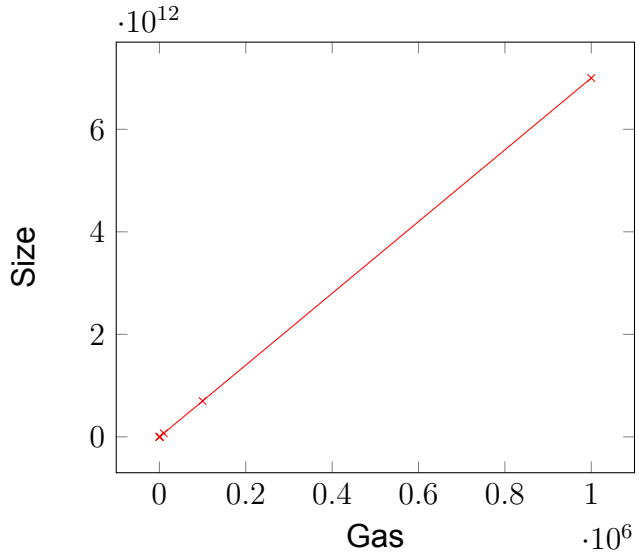


Figure 41: Data storage costs.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei

Table 8: Comparing methods of data storage.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei - Data Size: 32 bytes

Type	Size	Gas	USD
Data storage	32 bytes	34.916	$0.065
Hash pointer storage	32 bytes	34.850	$0.065
Data logging	32 bytes	25.995	$0.048
Hash pointer logging	32 bytes	25.995	$0.048

Table 9: Comparing methods of data storage.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei - Data Size: 1KB

Type	Size	Gas	USD
Data storage	1KB	724.730	$1.347
Hash pointer storage	32 bytes	34.850	$0.065
Data logging	1KB	104.310	$0.194
Hash pointer logging	32 bytes	25.995	$0.048

7.1.3 Zero Knowledge Proofs

A series of benchmarks are performed to measure the performance of the basic parts of the construction and verification of zero knowledge proofs. Proof generation is divided in three phases: `Setup`, `Compute` and `Verify`. `Setup` phase consists of the compilation of a `C`

Table 10: Comparing methods of data storage.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei - Data Size: 10KB

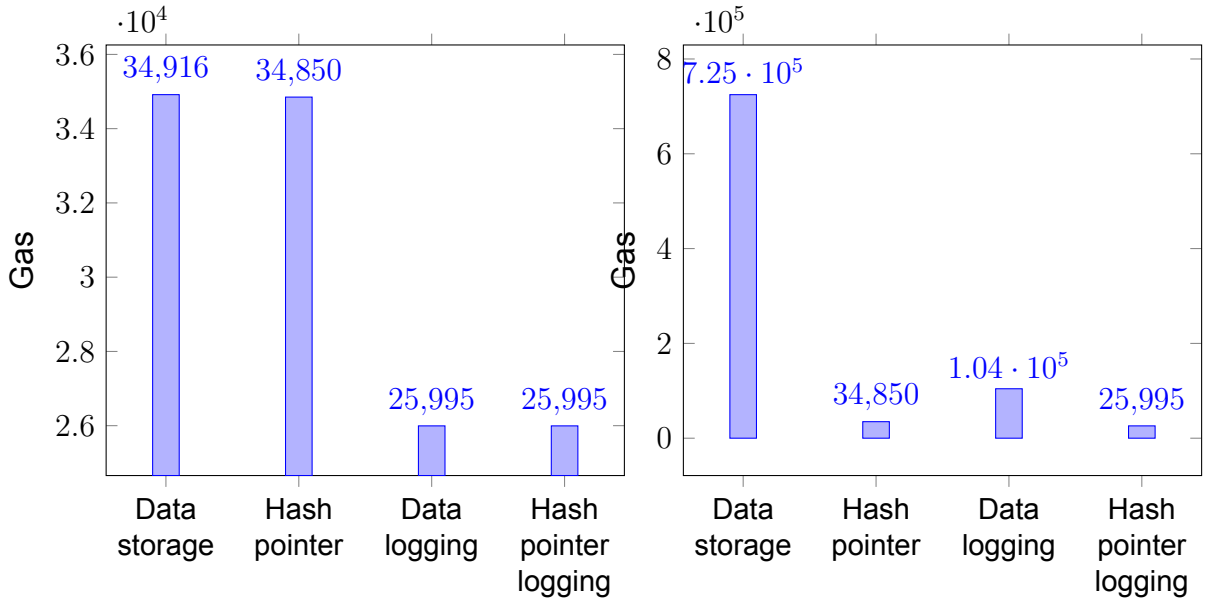
Type	Size	Gas	USD
Data storage	10KB	6.668.509	$12.39
Hash pointer storage	32 bytes	34.850	$0.065
Data logging	10KB	832.604	$1.547
Hash pointer logging	32 bytes	25.995	$0.048

Table 11: Comparing methods of data storage.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei - Data Size: 100KB

Type	Size	Gas	USD
Data storage	100KB	66.454.178	$123.472
Hash pointer storage	32 bytes	34.850	$0.065
Data logging	100KB	8.186.883	$15.211
Hash pointer logging	32 bytes	25.995	$0.048

Table 12: Comparing methods of data storage.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei - Data Size: 1MB

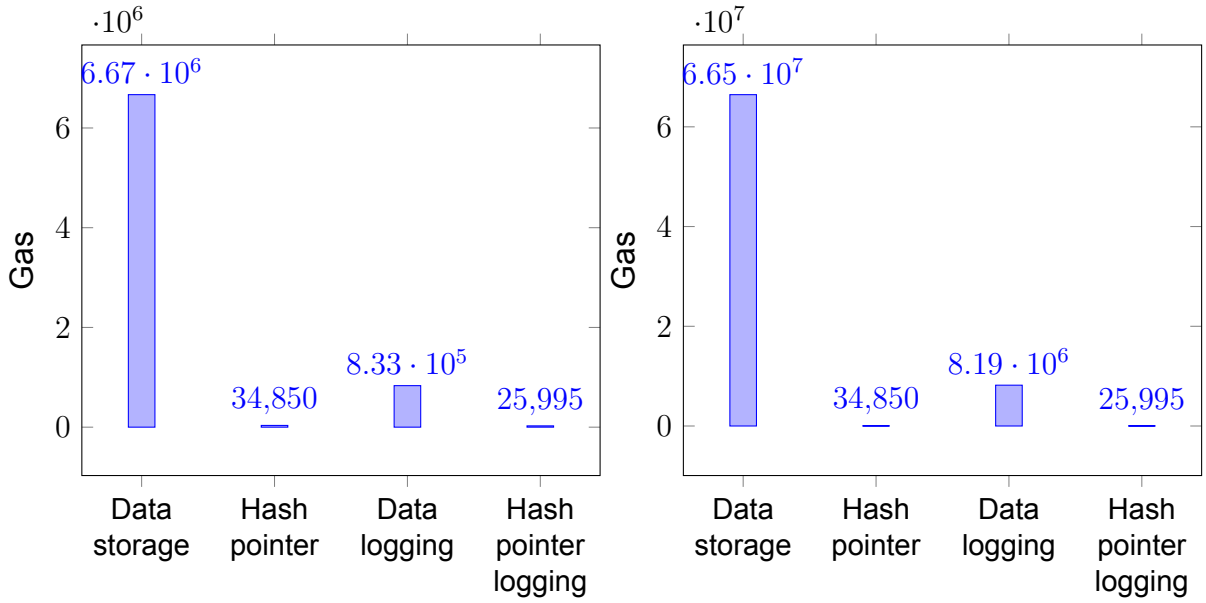
Type	Size	Gas	USD
Data storage	1MB	666.092.228	$1,237.599
Hash pointer storage	32 bytes	34.850	$0.065
Data logging	1MB	88.857.033	$165.096
Hash pointer logging	32 bytes	25.995	$0.048



(a) Size: 32 bytes **(b) Size: 1KB**

Figure 42: Storage methods

programm to QAP (§ 2.8.4) and the generation of the keys needed for proof construction and verification. `Setup` phase is executed once at application bootstrap and per processing algorithm. `Compute` phase consists of circuit evaluation, polynomial solving and proof generation with the use of the evaluation key. Lastly, `Verification` phase consists of proof verification by the verifier with the use of the verification key.



(a) Size: 10KB **(b) Size: 100KB**

Figure 43: Storage methods

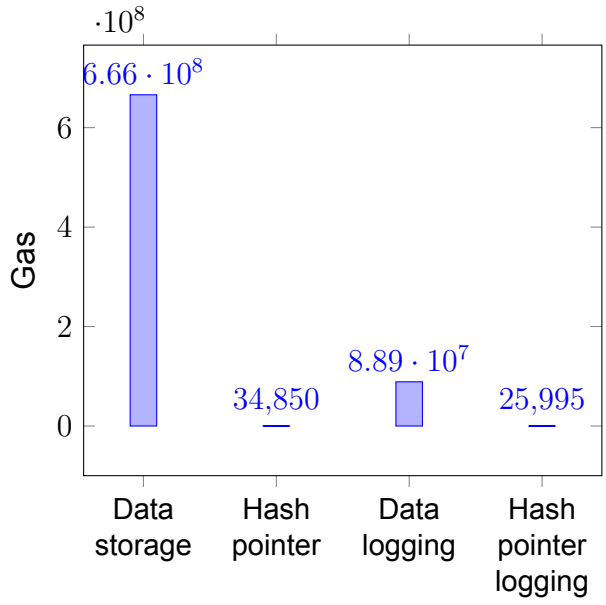


Figure 44: Storage methods - Size: 1MB

Table 13: Application Costs.
ETH Price: $942.99 (Feb 18, 2018) - Gas Price: 2 Gwei

Type	Gas	ETH	USD
App deployment	3.004.253	0.00601	$5.66
Register data set	233.487	0.000467	$0.44
Request for processing	89.206	0.0001784	$0.15
Register processor	83.274	0.0001665	$0.15
Notify processor	25.279	0.00005	$0.04

Tables 14 to 16 shows ZKP performance of the supported algorithms of the application. For each algorithm, we measure the time execution of each one of the three phases and the size of the generated keys and proofs. The keys are reasonably sized, with the eval-

uation key to typically range from 10 to 100 MB. The verification key is always constant size of 100KB. Due to the succinctness property of zkSNARKs the size of the proof is very small and constant for all programs regardless input size. The size of the evaluation keys and the execution time of each one of the phases are proportional to program's complexity.

Table 14: ZKP Performance: 100 values

	Setup (s)	Compute (s)	Verify (ms)	Eval Key (MB)	Ver Key (KB)	Proof (B)
Sum	10	8	28	0.016	100	304
Count	16	8	26	0.005	100	304
Max	17	9	28	1.7	100	304
Median	65	30	28	64	100	304
Min	17	9	51	1.7	100	304
Mean	16	8	19	0.054	100	304

Table 15: ZKP Performance: 1.000 values

	Setup (s)	Compute (s)	Verify (ms)	Eval Key (MB)	Ver Key (KB)	Proof (B)
Sum	10	8	27	0.144	100	304
Count	15	9	37	0.036	100	304
Max	29	14	34	17	100	304
Min	28	14	49	17	100	304
Mean	17	8	41	0.182	100	304
Median	1701	-	12	64	100	304

Table 16: ZKP Performance: 10.000 values

	Setup (s)	Compute (s)	Verify (ms)	Eval Key (MB)	Ver Key (KB)	Proof (B)
Sum	10	8	21	1.44	100	304
Count	17	8	32	0.335	100	304
Max	130	67	28	171	100	304
Min	126	67	27	171	100	304
Mean	22	9	27	1.5	100	304
Median	-	-	-	-	-	-

8. FUTURE WORK

8.1 Publicly verifiable zero knowledge proof

In the scheme presented in Chapter 5, only the requestor can verify the proof produced by the data processor crafted for her request. The output of the processing must be remain private. For that reason, is stored encrypted in the blockchain. As the output is a crucial input to the verification algorithm, only the requestor can validate the proof. This gives the requestor an opportunity to defy data processor's legitimacy and validity forcing the data processor to resort to a third part trusted entity that will enforce the data requestor to disclose the output of the process. To eliminate this, the verification algorithm must be run on the blockchain, where anyone can validate the proof, without revealing the output of the processing. The input of the verification algorithm must be encrypted. To achieve this, the proof generation algorithm must include the encryption procedure that encrypts the output. This ensures that the ciphertext corresponds to the output of the processing. Implementing the verification algorithm in the blockchain eliminates, also, the need of the salt (§ 5.5). The output is encrypted and an adversary cannot brute-force it efficiently.

8.2 Secure Multi party computation

In the setting of *multiparty computation* [19], sets of two or more parties with private inputs wish to jointly compute some (predetermined) function of their inputs [90]. Secure multiparty computation assume malicious behavior by a subset of participant entities.

The two important requirements of any secure computation protocol are *privacy* and *correctness*. The privacy requirement states that the parties should not learn anything else than the output of the computation and the correctness requirement states that each party should receive its correct output [90].

Informally, consider n parties with private inputs $x_1, x_2, \ldots, x_n$. The parties want to compute the outcome of the function $f(x_1, x_2, \ldots, x_n)$ where the respective inputs remain private.

The first secure multiparty computation problem was described by Yao in [175] and is called the *Yao's Millionaires' problem*. The problem discusses two millionaires, Alice and Bob, who are interested in knowing which of them is richer without revealing their actual wealth. Specifically, Yao's millionaires' problem is the problem of computing the predicate, $a \geq b$ where a is Alice's holding and b is Bob's holding, without disclosing anything more than the result to either party [97].

Multi-party computation protocols have a wide range of applications. They can be used in voting systems where each party votes for a candidate and they want to compute the winner of the voting without revealing their vote. Another use case it that of auctions. Several parties are bidding for a product where the winning party and maximum bid should be determined, without revealing bids of other parties

Most MPC protocols make use of *secret sharing*. A secret sharing scheme allows a value to be shared among n parties where some of the parts or all of them are needed in order to reconstruct the secret [99]. Usually, there is a threshold t so that at least t part of the secret are needed to reconstruct the original secret.

Using an MPC protocol based on secret sharing eliminates the need of symmetric key

exchange and dataset decryption and exposition (§ 5.8, § 5.9). Data queries can be computed in a distributed way with the use of an MPC cluster of data processors. In addition, datasets can be split between data processors without them having access to the data in its entirety [180]. This way, the need for a trusted processor is eliminated as the data processors do not have access to the unciphered dataset.

8.3 Fees

As discussed in 5.2.3 a malicious requestor can threat the system by overflowing the network with multiple requests aiming to prevent other request to be fulfilled. A scalable *payment system* could be used as a countermeasure to this type of attacks where a requestor pays a fixed price per byte processed. The price can decrease as the data size increases.

A payment system helps to prevent DDoS attacks as each request has a cost and the attack becomes more expensive as the requests are increasing. A rational malicious requestor has to significantly gain more than the cost of the attack to be profitable. Moreover, a payment system will incentivize the data processors to follow the protocol honestly and contribute to the prosperity of the network.

8.4 Privacy Preserving Queries

The algorithms (queries) the system supports are not privacy-preserving. The privacy of the individuals in the dataset must be protected with various privacy-preserving techniques such as differential privacy [61], k-anonymity [151] and l-diversity [4]. This techniques can be either applied by the data controllers before handing over the dataset to the processor or by the processors themselves where a requestor wants the de-identification of the dataset as a processing procedure.

8.5 Reputation system

A *decentralized reputation system* [116], resilient to Sybil attacks, could eliminate the need of a centralized trust model of a public key infrastructure (PKI), which relies on a trusted entity to authenticate, identify and verify the participants of the system. The users in such systems built trust relationships among them and measure trustworthiness in a setting where assets can be exchanged between them [116]. The benefits of such systems are *pseudonymity* of the participants, truly decentralization and the increasing trust for the system itself.

A data controller, whom its dataset are of bad quality or fabricated with malicious intents, could be identified by the users of the system and gain bad reputation. It is evident, that the users of the network should choose datasets where their owners have high trust. The same applies to data processors and even the requestor themselves.

8.6 Analytics

The activity of the system could be analyzed by various tools to measure the functionality and the prosperity of the system. To achieve that, blockchain should be indexed and be monitored constantly by an analytic node that stores in a database all the needed information. Various charts and results can be derived by such analysis. For example:

- Transaction chart
- Address growth chart
- Hash rate growth chart
- Block difficulty
- Pending transactions per minute
- Network transaction fees
- Total dataset processing requests
- Total datasets
- Total processors
- Total data controllers
- Most trusted processors or controllers
- Most used datasets
- Most used algorithms
- Top 10 charts

8.7 Consent

A key aspect that is missing from the current solution is the implementation of *dynamic consent* which can empower the data owners. With the use of smart contracts data access policies could be implemented where data controllers are obligate to comply to them. That way, data owners have total control over their data and can decide when and by whom their data are accessed.

Modeling dynamic consent in smart contracts should be carefully analyzed taking into account design issues that are related to smart contracts lifecycle, required state variables for storing contract's information, and access restrictions to those variables.

Neisse et al. [131] proposed the following three models for data accountability and provenance tracking that comply with the GDPR(§ 4.2):

1. Contract for a specific controller: A contract where the data subject creates a contract tailored for each data controller

2. Contract for specific data: A contract where each subject's data instance is shared among all data controllers
3. Contract for multiple data subjects: A contract where the data controller specifies how the data from all subjects are treated from the controller

The first contract is more adequate for sensitive data [131], such as health data, since a subject-controller relationship [13] is being created where each patient has a different contract for each controller accessing their data, providing fine grained access control and provenance information. On the other hand, it has the highest cardinality among the others two types [131]. The second contract suffers from direct linkability as a unique subject address is needed compromising patients' privacy [131]. Lastly, the third one has the lowest cardinality among the others but also has the lower level of customization due to the limited number of contract options [131].

9. RELATED WORK

9.1 Enigma

Enigma is a blockchain-based protocol where decentralized applications can be deployed. It combines a blockchain with an off-chain storage where data are stored and enables different parties through a peer-to-peer network to run computation on them in a privacy preserving manner. Data are stored in a *distributed hash-table* (DHT) that is accessible through the blockchain. Data are not stored directly in the DHT as they are private. The DHT stores only references to the data that are encrypted and stored on the client side. The blockchain is used as an access-control manager and a log system which manages data access and identity verification. For computation an optimize verifiable secure MPC is used between computational nodes of the network where queries run, without a trusted third party.

9.2 MedRec

MedRec is a decentralized record management system to handle *Electronic Health Records* (EHRs) with the use of the Ethereum blockhain. Its purpose is to address the various problems the health care industry faces especially when it comes to medical records exchange between various organizations. With the use of smart contracts a dynamic consent mechanism is created where patients form contracts with various providers that enforce them to follow the various access permissions the patient authorized. The medical records are stored on providers without any need to change their infrastructure. A gatekeeper node is installed in each provider and is responsible to listen to queries, that are predefined and crafted based on patient access permissions, and fulfill them. When a request for query execution arrives the provider checks the blockchain for access permissions and then it executes it locally and return the results.

9.3 Datum

Datum is a decentralized market place where users can share or sell their data. Data are encrypted and stored in a decentralized data store running by storing nodes. Storing nodes are paid by the users for their service in a token called DAT. Data, before leaving the client storage, are cleaned from personally identifying information and then saved to the network. Data consumers can request and buy datasets that are stored in the network. When a user accepts a buy request from a data consumer, it authorized it and the decryption key is sent to the data consumer which in turn pays the user with DAT. The procedure of data exchange is governed by a smart contract responsible for fairness in the system.

9.4 ADSNARK

Succinct Non-Interactive Arguments of Knowledge on Authenticated Data (AD-SNARK) is a cryptographic primitive in which privacy-preserving proofs on authenticated data can

be generated and verified. In a AD-SNARK setting a party is requested for a proof of computation over a dataset acquired from a trusted source who vouch for the quality and legitimacy of the data. The verifier is able to check the validity of the dataset authenticated by the source. The creators of ADSNARK are inspired by ZQL [77], a query language for expressing simple computations on private data. ADSNARK is achieved by embedding digital signatures into SNARKs. The data source, which is trusted, authorizes the data by producing a valid digital signature which is taken as public input to the proof generation algorithm.

10. CONCLUSION

In this work, we presented a data sharing and processing in a privacy preserving manner platform. Through the platform data controllers can register their datasets and make them available for processing. Data processors can process datasets on behalf of data requestors. Anyone can be a data requestor and request a data processing on a specific dataset and for a specific algorithm.

We use a distributed ledger as the controller of the system. The use of distributed ledger helps to unite trustless entities with shared interests – the data processing of sensitive datasets. Utilizing the blockchain as the coordinator of the system all actions of participants are recorded. Thus, trasparency, accountability, non-repudation, data provenance, and auditability is enabled. In addition to the blockchain, we use a Zero Knowledge verifiable computation scheme with which the data processors can produce a proof of correct computation without revealing the dataset itself. Requestors can verify proofs and be assured of the validity of the output.

We aim to a fully decentralized data sharing ecosystem where researchers and organization can extract all the valuable information of datasets without compromising privacy.

ABBREVIATIONS - ACRONYMS

SET	Secure Electronic Transaction
NIZK	Non-interactive zero knowledge
SNARK	Succinct Non-Interactive Argument of Knowledge
zkSNARK	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge
UTXO	Unspent transaction output
SHA	Secure Hash Algorithm
PoW	Proof of Work
PoS	Proof of Stake
PBFT	Practical Byzantine Fault Tolerance
EVM	Ethereum Virtual Machine
ASIC	Application-specific Integrated Circuit
MPC	Multi-party Computation
DHT	Distributed Hash Table
Dapp	Decentralised Application
GDPR	General Data Protection Regulation
DPR	Data Protection Directive
EU	European Union
Opcode	Operation Code
VC	Verifiable Computation
CLI	Command-line interface
PKI	Public Key Infrastructure
CRS	Common Reference String
QSP	Quadratic Span Program
QAP	Quadratic Arithmetic Program
R1CS	Rank 1 Constraint System
URI	Uniform Resource Identifier
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
UI	User Interface

SegWit	Segregated Witness
SPOF	Single Point of Failure
DoS	Denial-of-service
DDoS	Distributed denial-of-service
IoT	Internet of Things
DDH	Decisional Diffie Hellman
PPT	Probabilistic polynomial-time
PID	Persistent identifiers
ECC	Elliptic curves
OTP	One-time pad
PRG	Pseudorandom generator
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDH	Elliptic Curve Diffie–Hellman
DSA	Digital Signature Algorithm
NIST	Digital Signature Algorithm
DLGO	Discrete logarithm
CDH	Computational Diffie-Hellman
DDH	Decisional Diffie-Hellman
ECB	Electronic Codebook
CBC	Cipher Block Chaining
OFB	Output Feedback
CTR	Coutner
IV	Initial Vector
P2P	Peer-to-peer
IP	Internet Protocol
ETH	Ether
GHOST	Greedy Heaviest Observed Subtree
DNN	Deep Neural Network
EHR	Electronic Health Record
CCTV	Closed-circuit television
AD-SNARK	Succinct Non-Interactive Arguments of Knowledge on Authenticated Data

MAC	Message Authentication Code
OCB	Offset Codebook Mode
GCM	Galois/Counter Mode
CBC-MAC	Cipher Block Chaining Message Authentication Code

APPENDIX A. SOURCE CODE UNDER MIT LICENCE

MIT License

Copyright (c) 2017 Christos Nasikas

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the S̈oftware)̈, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ÄS IS,̈ WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

APPENDIX B. CREATIVE COMMONS ATTRIBUTION 4.0 INTERNATIONAL PUBLIC LICENSE

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License (”Public License”). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

B.1 Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter’s License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

B.2 Section 2 – Scope.

a. **License grant.**

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to: A. reproduce and Share the Licensed Material, in whole or in part; and B. produce, reproduce, and Share Adapted Material.
2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.
3. Term. The term of this Public License is specified in Section 6(a).
4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.
5. Downstream recipients.
   A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.
   B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.
6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. **Other rights.**

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
2. Patent and trademark rights are not licensed under this Public License.
3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

B.3 Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. **Attribution.**

1. If You Share the Licensed Material (including in modified form), You must:
   A. retain the following if it is supplied by the Licensor with the Licensed Material:
      i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated); a copyright notice;
      ii. a notice that refers to this Public License;
      iii. a notice that refers to the disclaimer of warranties;
      iv. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;
   B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and
   C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.
2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.
3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.
4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

B.4 Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

B.5 Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

B.6 Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or
   2. upon express reinstatement by the Licensor.

c. For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

d. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

e. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

B.7 Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

B.8 Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

REFERENCES

[1] TheDAO smart contract. https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413#code.

[2] Feross Aboukhadijeh. JavaScript standard style, 2018.

[3] Carlisle Adams, Steve Lloyd, and Carlisle Adams. *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley, Boston, 2nd ed edition, 2003.

[4] Charu C. Aggarwal and Philip S. Yu. *A General Survey of Privacy-Preserving Data Mining Models and Algorithms*, pages 11–52. Springer US, Boston, MA, 2008.

[5] Akasha. https://akasha.world/.

[6] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 443–458, Washington, DC, USA, 2014. IEEE Computer Society.

[7] A.M. Antonopoulos. *Mastering Bitcoin*. O'Reilly, 2014.

[8] The Mail Archive. Satoshiś posts to cryptography mailing list, 2008. [Online; accessed 2018].

[9] The Mail Archive. Bitcoin v0.1 released, 2009. [Online; accessed 2018].

[10] Rachel Arthur. Data is the new oil' and more from sxsw, 2016.

[11] Augur. http://www.augur.net/.

[12] Bootstrap Authors. Bootstrap: The most popular html, css, and javascript framework for developing responsive, mobile first projects on the web, 2018.

[13] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. MedRec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE, aug 2016.

[14] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On bitcoin and red balloons. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pages 56–73, New York, NY, USA, 2012. ACM.

[15] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[16] Backfeed. http://backfeed.cc/.

[17] Lear Bahack. Theoretical bitcoin attacks with less than half of the computational power (draft). *CoRR*, abs/1312.7013, 2013.

[18] Dr. Arati Baliga. Understanding blockchain consensus models. Technical report, Persistent Systems Ltd, 2017.

[19] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[20] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 90–108, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[21] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 781–796, San Diego, CA, 2014. USENIX Association.

[22] Juan Benet. Ipfs - content addressed, versioned, p2p file system. Technical report, 2014.

[23] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 421–439, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[24] Iddo Bentov, Alex Mizrahi, and Meni Rosenfeld. Decentralized prediction market without arbiters. In *Financial Cryptography Workshops*, 2017.

[25] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349, New York, NY, USA, 2012. ACM.

[26] Bitcoin. Bip141 — bitcoin improvement proposals, 2018. [Online; accessed 2018].

[27] I. Blake, G. Seroussi, N. Smart, Eugene Spafford Collection, London Mathematical Society, and N.J. Hitchin. *Elliptic Curves in Cryptography*. Lecture note series. Cambridge University Press, 1999.

[28] blockchain.info. Bitcoin blockchain size, 2018. [Online; accessed 2018].

[29] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, December 1991.

[30] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. `https://crypto.stanford.edu/~dabo/pubs/abstracts/bookShoup.html`, 2017.

[31] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, May 2015.

[32] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *In: Financial Cryptography and Data Security*, 2014.

[33] Sean Bowe, Ariel Gabizon, and Matthew D Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. Technical report, TR 2017/602, IACR, 2017.

[34] Johannes Buchmann. The digital signature algorithm (dsa). 2001.

[35] Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally efficient accountable time-stamping. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, pages 293–305, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[36] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

[37] Miles Carlsten, Harry Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 154–167, New York, NY, USA, 2016. ACM.

[38] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[39] Konstantinos Chalkias. Demonstrate how zero-knowledge proofs work without using maths — linkedin corporation, 2017. [Online; accessed 23-January-2017].

[40] David Chaum. *Blind Signatures for Untraceable Payments*, pages 199–203. Springer US, Boston, MA, 1983.

[41] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[42] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '88, pages 319–327, London, UK, UK, 1990. Springer-Verlag.

[43] Coinpoker. `https://coinpoker.com/`.

[44] Consensys. Ganache, 2018.

[45] Consensys. Truffle, 2018.

[46] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 106–125, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[47] Cryptokitties. https://www.cryptokitties.co/.

[48] Sophie Curtis. How much is your personal data worth? *The Telegraph*.

[49] Cypherpunks. Magic money digicash system — cypherpunks, 1994. [Online; accessed 2017].

[50] Wei Dai. b-money, 1998. *URL http://www. weidai. com/bmoney. txt*, 1998.

[51] Phil Daian. Analysis of the DAO exploit.

[52] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[53] Matt Day. Microsoft touts developer tools, business software at build, 2016.

[54] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International Conference on Financial Cryptography and Data Security*, pages 79–94. Springer, 2016.

[55] Maria Deutscher. Ibm's ceo says big data is like oil, enterprises need help extracting the value, 2013.

[56] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[57] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Untangling blockchain: A data processing view of blockchain systems. *CoRR*, abs/1708.05665, 2017.

[58] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM review*, 45(4):727–784, 2003.

[59] Joan Antoni Donet Donet, Cristina Pérez-Solà, and Jordi Herrera-Joancomartí. The bitcoin p2p network. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security*, pages 87–102, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[60] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[61] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[62] Cynthia Dwork and Moni Naor. *Pricing via Processing or Combatting Junk Mail*, pages 139–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[63] Billy Ehrenberg. How much is your personal data worth? *The Guardian*.

[64] Khaled El Emam, Luk Arbuckle, Gunes Koru, Benjamin Eze, Lisa Gaudette, Emilio Neri, Sean Rose, Jeremy Howard, and Jonathan Gluck. De-identification methods for open health data: the case of the heritage health prize claims dataset. *Journal of medical Internet research*, 14(1), 2012.

[65] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, Jul 1985.

[66] Emily Cadman Emily Steel, Callum Locke and Ben Freese. How much is your personal data worth? *Financial Times*.

[67] Ethereum. web3.js: Ethereum javascript api, 2018.

[68] Etheroll. https://etheroll.com/.

[69] Bitcoin Block Explorer. Bitcoin block 0, 2009. [Online; accessed 2018].

[70] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243, 2013.

[71] Cardano Foundation. Cardano — cardano foundation, 2017. [Online; accessed 2017].

[72] Ethereum Foundation. Ethash, 2017. [Online; accessed 12-October-2017].

[73] Ethereum foundation. Design rationale. Technical report, 2018.

[74] Ethereum Foundation. The solidity contract-oriented programming language, 2018.

[75] Node.js Foundation. Express: Fast, unopinionated, minimalist web framework for node.js, 2017.

[76] Node.js Foundation. Node.js, 2018.

[77] Cédric Fournet, Markulf Kohlweiss, George Danezis, Zhengqin Luo, et al. Zql: A compiler for privacy-preserving data processing. In *USENIX Security Symposium*, pages 163–178, 2013.

[78] Patrick Gallagher. Digital signature standard (dss). *Federal Information Processing Standards Publications, volume FIPS*, pages 186–3, 2013.

[79] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[80] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 626–645, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[81] Gnosis. `https://gnosis.pm/`.

[82] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[83] P. Golodoniuc, N.J. Car, and J. Klump. Distributed persistent identifiers system design. *Data Science Journal*, 16:34, 2017.

[84] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. *Journal of Cryptology*, 27(3):506–543, 2014.

[85] Bitfury Group. On blockchain auditability. Technical report, Bitfury Group, 2016.

[86] BitFury Group and J. Garzik. Public versus private blockchains. part 1: Permissioned blockchains. Technical report, BitFury Group, 2016.

[87] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, Jan 1991.

[88] D. Hankerson, A.J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer New York, 2006.

[89] Michael Haupt. “data is the new oil”—a ludicrous proposition, 2016.

[90] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Springer Berlin Heidelberg, New York, NY, USA, 1st edition, 2010.

[91] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin’s peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, Washington, D.C., 2015. USENIX Association.

[92] Jordi Herrera-Joancomartí. Research and challenges on bitcoin anonymity. In Joaquin Garcia-Alfaro, Jordi Herrera-Joancomartí, Emil Lupu, Joachim Posegga, Alessandro Aldini, Fabio Martinelli, and Neeraj Suri, editors, *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 3–16, Cham, 2015. Springer International Publishing.

[93] Blockchain Hub. Blockchains and distributed ledger technologies — blockchain hub, 2017. [Online; accessed 2017].

[94] Eric Hughes. A cypherpunḱs manifesto — eric hughes, 1993. [Online; accessed 2017].

[95] Facebook Inc. React: A javascript library for building user interfaces, 2018.

[96] Ecma International. Ecmascript® 2015 language specification, 2018.

[97] I. Ioannidis and A. Grama. An efficient protocol for yao’s millionaires’ problem. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, pages 6 pp.–, Jan 2003.

[98] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[99] P. Kamm and L. Laud. *Applications of Secure Multiparty Computation*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2015.

[100] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.

[101] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[102] Aggelos Kiayias. Cryptography — primitives and protocol. Based on notes by S. Pehlivanoglu, J. Todd, K. Samari, T. Zacharias and H.S. Zhou.

[103] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 61–78, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[104] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work, 2017.

[105] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. *Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol*, pages 357–388. Springer International Publishing, Cham, 2017.

[106] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. Cryptology ePrint Archive, Report 2015/675, 2015. https://eprint.iacr.org/2015/675.

[107] RJ Krawiec. Blockchain: Opportunities for health care. Technical report, Deloitte Conslulting LLP, 2016.

[108] Stephan Kudyba and Matthew Kwatinetz. Introduction to the big data era. *Big Data, Mining, and Analytics*, pages 1–17, 2014.

[109] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 30–41, New York, NY, USA, 2014. ACM.

[110] Meglena Kuneva. Roundtable on online data collection, targeting and profiling, 2009.

[111] SCIPR Lab. libsnark: a c++ library for zksnark proofs, 2018.

[112] Protocol Labs. Filecoin: A decentralized storage network. Technical report, 2017.

[113] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[114] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, pages 446–465, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[115] Kevin Liao and Jonathan Katz. Incentivizing blockchain forks via whale transactions. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security*, pages 264–279, Cham, 2017. Springer International Publishing.

[116] Orfeas Stefanos Thyfronitis Litos and Dionysis Zindros. Trust is risk: a decentralized financial trust platform. In *International Conference on Financial Cryptography and Data Security*, pages 340–356. Springer, 2017.

[117] DataStreamX Pte Ltd. Datastreamx, 2018.

[118] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.

[119] Laurens Van Houtven (lvh). Crypto 101. https://www.crypto101.io/, 2017.

[120] Hartwig Mayer. zk-snark explained: Basic principles. 2016.

[121] Erika McCallister, Timothy Grance, and Karen A Scarfone. Guide to protecting the confidentiality of personally identifiable information (pii). Technical report, 2010.

[122] Adrian McCullagh and William Caelli. Non-repudiation in the digital environment. *First Monday*, 5(8), 2000.

[123] Declan McCullagh. Verizon draws fire for monitoring app usage, browsing habits, 2012.

[124] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 127–140, New York, NY, USA, 2013. ACM.

[125] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.

[126] Metamask. `https://metamask.io/`.

[127] MHMD. My health my data (mhmd), 2018. [Online; accessed 2018].

[128] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[129] Arvind Narayanan. What happened to the crypto dream?, part 1. *IEEE security & privacy*, 11(2):75–76, 2013.

[130] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, Princeton, NJ, USA, 2016.

[131] Ricardo Neisse, Gary Steri, and Igor Nai Fovino. A blockchain-based approach for data accountability and provenance tracking. *CoRR*, abs/1706.04507, 2017.

[132] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future Internet*, 5(2):237–250, 2013.

[133] K. J. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. In *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pages 280–285, June 2014.

[134] Official Journal of the European Union (1995). Directive 95/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, 1995.

[135] Official Journal of the European Union (4 May 2016). Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, 2016.

[136] Michael Palmer. Data is the new oil, 2006.

[137] Alisa Pankova. Succinct Non-Interactive Arguments from Quadratic Arithmetic Programs. Technical report, University of Tartu, Cybernetica AS, December 2013.

[138] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2013.

[139] Michael Parsons. Cardano: A blockchain with privacy and regulation — cardano foundation, 2017. [Online; accessed 2017].

[140] Pepper Project. Pequin: An end-to-end toolchain for verifiable computation, snarks, and probabilistic proofs, 2018.

[141] q Data Exchange. q data exchange, 2018.

[142] Minghua Qu. Sec 2: Recommended elliptic curve domain parameters, 1999.

[143] Jean-Jacques Quisquater, Louis Guillou, Marie Annick, and Tom Berson. How to explain zero-knowledge protocols to your children. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 628–631, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[144] Elizabeth D Rather, Donald R Colburn, and Charles H Moore. The evolution of forth. In *ACM Sigplan Notices*, volume 28, pages 177–199. ACM, 1993.

[145] Redux. Redux: Predictable state container for javascript apps, 2018.

[146] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 1318–1326, Oct 2011.

[147] Christian Reitwießner. zksnarks in a nutshell. Technical report, 2016.

[148] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security*, pages 6–24, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[149] Tim Ruffing and Pedro Moreno-Sanchez. Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security*, pages 133–154, Cham, 2017. Springer International Publishing.

[150] Sara Sabour, Yanshuai Cao, Fartash Faghri, and David J Fleet. Adversarial manipulation of deep representations. *arXiv preprint arXiv:1511.05122*, 2015.

[151] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, 1998.

[152] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, May 2014.

[153] Amitabh Saxena, Janardan Misra, and Aritra Dhar. Increasing anonymity in bitcoin. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security*, pages 122–139, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[154] Bruce Schneier and Chris Hall. An improved e-mail security protocol. In *Computer Security Applications Conference, 1997. Proceedings., 13th Annual*, pages 227–230. IEEE, 1997.

[155] K Schwab, A Marcus, JO Oyola, W Hoffman, and M Luzi. The emergence of a new asset class. Technical report, An Initiative of the World Economic Forum, 2011.

[156] ScienceDaily. Big data, for better or worse: 90% of world's data generated over last two years, 2013.

[157] C. E. Shannon. Communication theory of secrecy systems*. *Bell System Technical Journal*, 28(4):656–715.

[158] Yonatan Sompolinsky and Aviv Zohar. *Secure High-Rate Transaction Processing in Bitcoin*, pages 507–527. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[159] Peter Sondergaard. Gartner says worldwide enterprise it spending to reach $2.7 trillion in 2012, 2011.

[160] E. Stark, M. Hamburg, and D. Boneh. Symmetric cryptography in javascript. In *2009 Annual Computer Security Applications Conference*, pages 373–381, Dec 2009.

[161] Dawex Systems. Dawex, 2018.

[162] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 1996.

[163] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[164] IBM Global Business Services Public Sector Team. Blockchain: The chain of trust and its potential to transform healthcare – our point of view. Technical report, IBM, 2016.

[165] N. Thiranant, Y. S. Lee, and H. Lee. Performance comparison between rsa and elliptic curve cryptography-based qr code authentication. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 278–282, March 2015.

[166] Richard Titus. Data is the new oil, 2006.

[167] Nicolas van Saberhagen. Cryptonote v 2.0. Technical report, Persistent Systems Ltd, 2013.

[168] Serge Vaudenay. Security flaws induced by cbc padding—applications to ssl, ipsec, wtls... In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 534–545. Springer, 2002.

[169] Marko Vukolić. *The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication*, pages 112–125. Springer International Publishing, Cham, 2016.

[170] L.C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Discrete Mathematics and Its Applications. CRC Press, 2003.

[171] Osamu Watanabe and Osamu Yamashita. *An improvement of the digital cash protocol of Okamoto and Ohta*, pages 436–445. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[172] Wikipedia. Open data — Wikipedia, the free encyclopedia, 2017. [Online; accessed 12-January-2018].

[173] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, James Prestwich, Gordon Hall, Patrick Gerbes, Philip Hutchins, and Chris Pollard. Storj: A peer-to-peer cloud storage network. Technical report, 2016.

[174] Dr. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2017.

[175] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[176] Ben Yuan, Wendy Lin, , and Colin McDonnell. Blockchains and electronic health records. Technical report, 2017.

[177] zcash. What are zk-snarks? [Online; accessed 2018].

[178] Dionysios Zindros. Trust in decentralized anonymous marketplaces. 2016.

[179] Aviv Zohar. Bitcoin: Under the hood. *Commun. ACM*, 58(9):104–113, August 2015.

[180] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *CoRR*, abs/1506.03471, 2015.

[181] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. Decentralizing privacy: Using blockchain to protect personal data. *2015 IEEE Security and Privacy Workshops (SPW)*, 00:180–184, 2015.