UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Cunha Reis

Extending Smart Contracts security through cryptographic protocols

Uberlândia, Brasil

2020

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Cunha Reis

Extending Smart Contracts security through cryptographic protocols

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Ivan Sendin

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2020

Matheus Cunha Reis

Extending Smart Contracts security through cryptographic protocols

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 21 de dezembro de 2020:

Ivan Sendin
Orientador

Maria Adriana Vidigal de Lima
Professor

Rodrigo Sanches Miani
Professor

Uberlândia, Brasil
2020

Resumo

Smart Contracts juntamente com a Blockchain trouxeram um novo conjunto de possibilidades no desenvolvimento de protocolos de segurança: os usuários ganham garantia da execução correta das etapas dos protocolos enquanto a Blockchain proporciona um ambiente seguro e imutável para a execução desses protocolos. Em contraste, as atuais soluções baseadas em Smart Contracts não fornecem a privacidade de dados necessária para os participantes interessados em um acordo, especialmente quando não há confiança mútua entre eles, algo indesejável em cenários de comércio eletrônico. Para fornecer uma solução para compra de bens digitais onde as partes envolvidas não confiam entre si, este trabalho apresenta um protocolo de comércio justo usando Smart Contracts, Bloom Filters e a plataforma Ethereum. Em geral, o protocolo provou ser promissor para negócios de médio a alto custo. Entretanto, seu uso em cenários de baixo custo deve ser abordado com cautela, pois os custos da execução do protocolo devem ser levados em conta.

Palavras-chave: Blockchain, Smart Contracts, Ethereum, Fair Trade.

Abstract

Smart Contracts along with Blockchain bring a new set of possibilities in the security protocol development: participants gain guarantees of the correct execution of the protocol steps while Blockchain provides a secure and immutable environment to execute those protocols. In contrast, current Smart Contract solutions do not provide the necessary data privacy for the interested participants in a trade, especially when they are not mutually trustworthy, being awful in scenarios of *e-goods* commerce. To provide a solution for *e-goods* trading where the involved parties do not trust each other, this work presents a fair trade protocol using Smart Contracts, Bloom Filters, and the Ethereum platform. In general, the protocol proved to be promising for medium to high-cost trades. However, its use in small-cost scenarios must be approached with caution, as the protocol's cost execution must be taken into account.

Keywords: Blockchain, Smart Contracts, Ethereum, Fair Trade.

List of Figures

Listings

List of Tables

List of abbreviations and acronyms

P2P	*Peer-To-Peer*
ZKP	*Zero-Knowledge Proof*
SMC	*Security Multi-Part Computation*
MPC	*Multi-Part Computation*
DHT	*Distributed Hash Table*
IT	*Information Technology*
PSI	*Private Set Intersection*
DAPPS	*Decentralized Applications*
EOA	*Externally Owned Account*
POW	*Proof of Work*
DoS	*Denial of Service*
EVM	*Ethereum Virtual Machine*

Contents

1 Introduction

At the end of 2008, Satoshi Nakamoto, a pseudonym for an individual or a group of unknown people, published the article *Bitcoin: A peer-to-peer electronic cash system* (NAKAMOTO, 2008) introducing an electronic cash system that uses a *Peer-To-Peer* (P2P) network, which allows a person transfer money to another without depending on a central authority or financial institution. This system called *Bitcoin*, also defined as **digital money ecosystem** (ANTONOPOULOS, 2014) conciliated many tools and technologies already known in a fast, secure, immutable and decentralized structure.

In the Bitcoin Protocol, units of currency also called *Bitcoin* are used to store and perform transactions, either to transfer amounts to users or organizations or to buy and sell products. After concluding the transactions, they are grouped and saved in blocks, consequently chained in a list termed *ledger*, where financial transactions of users are stored and distributed to members of the P2P network, without dependence and interference of third parties, ensuring the veracity of data.

Blockchain or *Distributed Ledger Technology* (DLT) are another names given to *ledger*, that could be described as the technology in the heart of the Cryptocurrencies. This distributed database system keeps the blocks of the transactions sequentially, protecting and encapsulating with a series of cryptographic methods, accessible for the world with permanent and verifiable data.

Definition 1.1. ***Blockchain*** *At a technical level, blockchain can be defined as an immutable ledger for record transactions, maintained within a distributed network of mutually untrusting peers (GAUR et al., 2018).*

Definition 1.2. ***Distributed Ledger Technology (DLT)*** *A ledger that is distributed among its participants and spread across multiple sites or organizations where the records are stored contiguously (BASHIR, 2017).*

Although used mainly as a *ledger* in Cryptocurrencies, it was noted that the Blockchain's applicability was not restricted at this. Many projects were written and developed to explore new possibilities for the *Blockchain*, using it for storage mechanisms and financial transactions processing, including in the areas of health, food, and music industry, among others (MATTILA, 2016).

One of the main applications and most used is Ethereum, a decentralized and open-source distributed computing platform that provides uploading and autonomous execution of scripts. The Blockchain is used to synchronize and store the changes of system states, along with a cryptocurrency called Ether utilized to control the system,

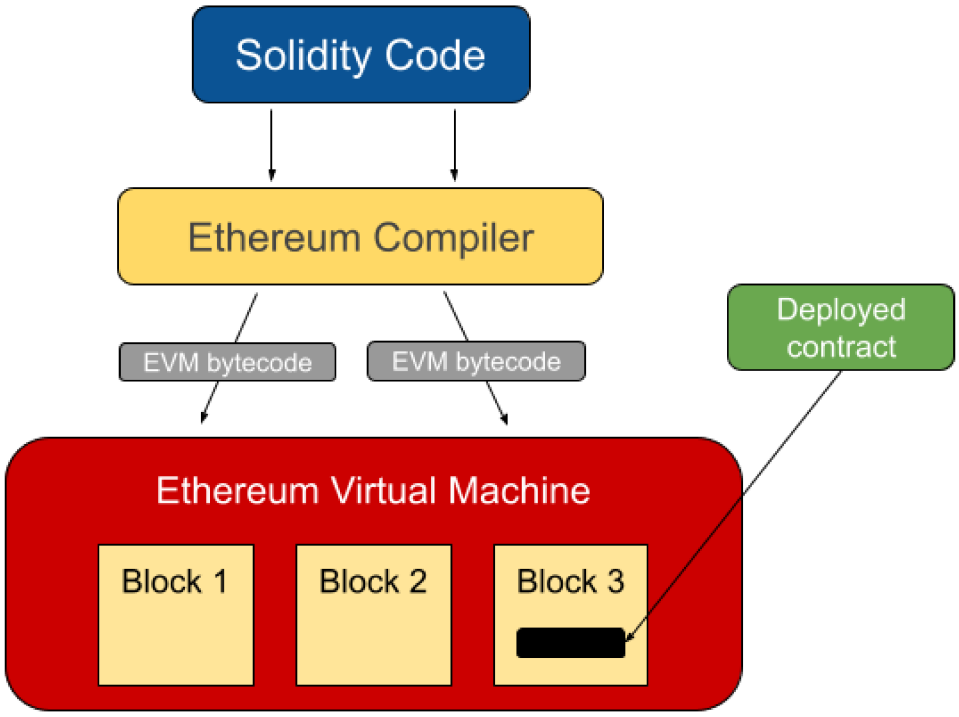
measuring and restricting the costs of execution resources (ANTONOPOULOS; WOOD, 2018; ROBINSON, 2018).

Similarly Bitcoin, the *Ether* is also used in transferences or buy/sell transactions. However, it can be used in the *Ethereum* ecosystem to execute functions, create contracts, or perform standard transactions.

Those *scripts* known as well as *Smart Contracts* are described as a set of promises, specified in digital form, that includes protocols within which the parties perform on these promises (SZABO, 2018). In other words, they are programmable smart protocols that execute the terms of an established contract between multi parts when their conditions are fulfilled. Among its characteristics, it is essential to highlight the fact that they are irreversible, trustable, decentralized, and public, meaning that the code and data related to contracts could be read for any person (BUTERIN, 2014).

For the development of contracts, it is necessary to use a programming language (Go, Javascript, Solidity, C++, etc.) to develop contracts, being Solidity the most used. It is a modified and optimized language according to Ethereum's limitations. By the time a contract is ready to be published, Ethereum takes care of compiling the code into bytecode and implements an isolated virtual machine called Ethereum Virtual Machine (EVM) to run those bytecodes (Figure 1) (SOLIDITY, 2019).

Figure 1 – The architecture of Ethereum



Adapted from: https://www.edureka.co/blog/ethereum-tutorial-with-smart-contracts/

Besides Ethereum, there are other platforms, such as NEO (NEO, 2017) and EOS (LARIMER, 2017), which their specifications also allow the use of smart contracts. Due

to *Smart Contracts* being public and irreversible, contracts tend to be more susceptible to failures and security attacks. After all, people with malicious intentions can see and exploit the human factor's security flaws, being data, and money theft possible.

Since *Smart Contracts* are always associated with money, cryptocurrencies, and *tokens*, users are very concerned about confidentiality. Consequently, data on Blockchain is public, so it is impossible to hide pieces of information, including account balances, which can be discouraging and uninteresting to companies and users who have sensitive and confidential information.

Definition 1.3. ***Tokens*** *Cryptocurrencies built by the Smart contracts over the Ethereum's Platform, mainly used as part of an ecosystem or collaborative fundraising.*

1.1 Objectives

The main objective of this work is to analyze and provide solutions for improve the confidentiality and privacy in *Smart Contracts* environments, using various cryptographic methods and platforms in order to propose a new secure, private and trustable platform.

1.2 Methods

As previously discussed, Blockchain along with Ethereum and Smart Contracts have the lack of privacy and confidentiality. To try to solve some of these problems, there were created cryptographic methods that try to bring a bit more of confidentiality to Smart Contracts.

1.2.1 Zero-Knowledge Proof (ZKP)

Zero-Knowledge Proof is a cryptographic protocol that adds a new level of security and privacy to the Blockchain. It consists of proving that given a problem, and two parties, one party (the prover) can prove to another party (the verifier) that he knows how to solve the problem without conveying the solution itself, where the verifier can check if the prover knows how to solve the problem without knowing the solution. As an example, imagine have a color-blind friend and two balls, one red and another green, both of the same size, and you wanna prove to your friend that those balls have different colors. The friend as a color-blind person will certainly doubt your word. To make him believe, you could ask him to get the balls and hide them from you in his backs. Thenceforward, your friend with one ball in each hand will choose to change the hands or not, asking you if the balls were changed in each turn. He would think that is just coincidence or lucky in the first turns, but after some time, he will accept the proof that the balls have different colors (KOENS; RAMAEKERS, 2017).

This experiment implements the core strategy of ZKP applications. A true Zero-knowledge proof needs to have three different properties:

1. **Completeness**: If the information is true, the verifier should be convinced without any extra help;
2. **Soundness**: If the information is false, the verifier can not be convinced in any scenario;
3. **Zero-knowledge**: The verifier should not know any more information.

It should be noted that this algorithm works iteratively, where the prover keeps sending proofs to the verifier until he is satisfied. This iterable algorithm, also called interactive ZKP, does not work well in P2P environments, as it is impractical to keep checking the proof between nodes.

Due to this problem, the zkSnarks (REITWIESSNER, 2016) was created, a non-interactive algorithm where the prover sends the proof to the verifier and the verifier checks the proof itself without sending anything back to the prover which eliminates the iterable part of the algorithm, making it suitable for use in P2P networks.

1.2.2 Security Multi-Party Computation

Security Multi-Party Computation (SMC), also known as Multi-Party Computation (MPC), is an essential topic in the cryptographic world. SMC allows multiple parties to securely compute a function on their private inputs in a way that no data beyond the agreed would be available to the parties. Unlike other cryptographic methods, SMC does not try to protect the data from outside parties, but the attention is to keep secret the individual data of the connected parties (EVANS; KOLESNIKOV; ROSULEK, 2018).

As an example, let us imagine that we have a number of N parties interconnected $(p_1, p_2, ..., p_n)$, each of them with their respective data $(d_1, d_2, ..., d_n)$, and they want to compute the value of a public function passing their private data without revealing the data to other parties. The first and easier strategy would have an external trusted party (p_e), where the parties would reveal their data to p_e, and p_e would call the public function returning just the result for all the parties, assuring that two different parties would never have the data of each other. The problem with this approach is the solution itself. Just figure out if this trusted party decided not to be honest anymore and distribute the data of p_1 to p_2, the process would not be fair, and p_2 would always have an advantage against the other parties.

The main reason why the SMC protocol was developed is precisely to solve this problem. In the protocol, the parties' data are partitioned and distributed between all

parties, where each party has a meaningless small piece of data. To execute the public function, each party sends their data parts, and the merge of the data passed to the function gives a correct result without revealing any information.

The basic properties to be maintained by a multi-party protocol are:

Input privacy No information on the private data held by the parties may be inferred from the messages sent during the execution of the protocol. The only information that can be inferred from private data is anything that could be inferred from seeing the output of the function on its own;

Completeness Any appropriate subset of adverse parties willing to share data or deviate from orders during the implementation of the procedure should not be able to force honorable parties to achieve an inaccurate outcome. This correctness goal comes in two ways: either honest parties are guaranteed to calculate the correct output, or they abort if they find an error.

Examples of this protocol include privacy-preserving decision-making on distributed or financial medical data, privacy-preserving machine learning, auctions, online poker, private set intersection of sets belonging to various organizations, etc.

1.2.3 Enigma

A decentralized computation platform to build end-to-end decentralized applications without a trusted third party (ZYSKIND; NATHAN; PENTLAND, 2015). Enigma has two main properties:

1. **Private**: Enigma uses a multi-party security computation, splitting data between different nodes and merging them when a computation is needed, assuring that parties know their own data and nothing more;

2. **Scalable**: Unlike blockchains, the computation and data are not replicated by every node in the network, where Enigma creates different groups of nodes and attributes to them responsibility for different parts of the data. Assuring less use of storage and more computation's performance.

Designed for connecting to an existing Blockchain, the approach of Enigma consists of separating the code of a Smart Contract into two parts: public and private. The public part would be processed by the Blockchain itself and contain data and part of the code that can be visible for everyone. Whereas the private part and the computation would be off-loaded to the Enigma's off-chain and cannot be visible for anyone.

Definition 1.4. ***Distributed Hash Table*** *A distributed hash table (DHT) is a decentralized storage system that provides lookup and storage schemes similar to a hash table, storing key-value pairs.*

The code's execution is decentralized but not distributed, so the nodes execute the function redundantly to achieve the same state in all nodes. Beyond that, with an off-chain network Enigma solves known issues that the Blockchain cannot handle:

1. **Storage**: Solving the problem of the high cost of storage in the Blockchain, Enigma has a decentralized off-chain *Distributed Hash Table (DHT)* accessible through the Blockchain, which stores the link to the information, but not the data itself;
2. **Privacy-enforcing computation**: Enigma ensures correct execution of code without leak any private information for other nodes;
3. **Heavy processing**: As the computation's cost in Blockchain is expensive, Enigma pass through that executing the heavy computations off-chain and just distributing the results for the nodes.

1.2.4 Hawk

A framework for building privacy-preserving Smart Contracts. As Enigma, Hawk separates the code into the public and private portions. The private portion is protected along with the participant's data and the money exchanges, whilst the public portions do not touch private data or money.

Hawk owns his self compiler, which gets the public and private portions of code and generates jointly with cryptographic methods, the blockchain's program which will be executed by all consensus nodes, a program to be executed by users, and a program to be executed by a special party called the manager.

This manager can see the user's inputs and is trusted not to disclose the user's private data. Unlike the manager in common trusted third parties, the Hawk manager can not affect the execution of the contract or give some advantage to one of the parties. His role consists of performing the multiparty computation and guaranteeing the right execution of Smart Contracts (KOSBA et al., 2016).

With all of those concepts, Hawk brings three properties as security guarantees:

1. **Input independent privacy**: Each user can not see other users' data before interacting with the contract. In other words, the data are independent;
2. **Posterior Privacy**: As the manager does not expose any data, the privacy of the data stored through transactions is guaranteed;

3. **Security against a dishonest manager**: The manager can not affect the system or have misbehaving acts without receiving high penalizations.

2 Theoretical Background

In this chapter, a theoretical background is built in order to explain and contextualize the reader about all theoretical concepts used in the present work.

2.1 Cryptographic Protocols

2.1.1 Commitment schemes

In cryptography, a commitment scheme is a primitive that allows a user to commit to a chosen value while keeping it secret to other users, including the capacity to reveal the committed value later. To understand, we could compare a commitment scheme with a safe, where a person Alice, locks an item inside the safe and gives it to another person Bob, without telling the safe's secret. The item inside the safe is hidden from Bob, who cannot open or change it cause he does not have the secret, but even without seeing the item, Bob can assure that the item did not change since he received the safe, no mattering if Alice reveals the key or not (BLUM, 1981).

Every commitment scheme has two phases:

1. **Commit:** The phase where a value is chosen and specified;
2. **Reveal:** The phase where the chosen value at the commit phase is revealed and specified.

Moreover, it needs to follow two properties:

- **Hiding:** The property that guarantees the secrecy and privacy of the choices in both phases;
- **Biding:** The property that certificates the immutability of the values after the commitment phase.

Generally, on simpler commitment schemes, in the commitment phase, a sender delivers a message to the recipient, as long as that message cannot be recognized by him, which is called *commitment*. Whereas in the reveal phase, the recipient calculates and validates the message sent in the commitment phase, which is known by *opening*.

Nowadays, several applications require values to be hidden, and commitment schemes are excellent solutions for this kind of problem. Some of the most known applications are:

- **Coin Flipping:** A coin flip protocol when the participants, Alice and Bob, are not physically in the same place. In a normal game, Alice and Bob would bet in one of each face of the coin and flip it to check the winner, but that does not work when the parties are not in the same place, seeing that the first to start the game would have to bet a coin's face and send it to the other one and the second player could cheat adapting the result of the flip to win.

  By commitment schemes, a secure protocol can be done in five steps:

  1. Alice chooses her bet, hashes it with a secret key and send this hash as a commitment to Bob;
  2. Bob flips the coin and exposes the result;
  3. Alice reveals the private key and her original bet to Bob;
  4. Bob verifies Alice's commit by comparing it to the hash of original bet using the secret key passed by her;
  5. If Alice's commit is correct and equal to the coin result, she wins. Else, Bob wins.

  The only way that Bob can distort the results is if he could break the hash function. So the security does not stand in the protocol itself but in the hash function used in it;

- **Blockchain:** As stated before, Zero-Knowledge Proof (ZKP) is a commitment scheme where a person could prove to another that he is capable of solving a specific problem without knowing the solution itself. In Blockchain, ZKP is regularly used to help in the validation of transactions, considering that the transaction's and users' data need to be calculated without identification;

- **Blind Auction:** A blind auction is a type of auction, where the bidders submit their bids simultaneously in secret so that no one would know how much the others have bid until the end of bidding time. The winner of the auction is revealed when the bids are exposed, and the bidder with the highest bid gains the prize.

  A commitment scheme to solve this problem uses the same concept of the Coin Flipping problem. Supposing that Alice is the auctioneer and Bob and Steve are the bidders, the protocol can be described in three phases:

  1. In the first phase, Bob and Steve will generate or choose a secret key for them. With those keys in their hands, they will concatenate it to their bid and hash it. The hash of the concatenation is what they will send to Alice to fulfill their bid;

2. After the bidding time expires, Alice will ask the participants (Bob and Steve) to send a proof of their bids. And so on, each one of them will send the original bid and the secret key used to hash it. Alice will get all the proofs and check the initial bids' veracity, revealing the correct ones. Subsequently, Alice reveals the winner by comparing the valid bids;
3. And in the last phase, the winner bidder gives the value of the bid to Alice and receives his prize.

2.1.2 Verifiable Computation

Cloud computing had become an essential tool in the IT industry, as it offers users computing resources without thinking about how these resources are provided and handled. Along with the high consumption of those cloud servers, which are typically managed by third-party vendors, the concerns about security, privacy, and correctness execution raises considering that the trust in third parties guarantees those concerns (LAI et al., 2014).

The correctness execution of the computation is one of the most worrying topics. It has been studied for a long time since if the server gives incorrect results, the users should be warned that the result data is not reliable. Users should also not have to trust third parties blindly, but the parties should gain the user's trust or prove that the algorithms will always be executed correctly and honestly in their platforms.

Thusly, verifiable computing or verified computation is defined as a protocol that allows a verifier (client) to send data to a provider (server) with the expectation that the result of a computation performed by him is verifiable without the need for recomputing data and provides a quick verification compared to running the algorithm itself.

There are diverse implementations of verifiable computation, and each one of them was created to solve special cases (LAI et al., 2014). Also, the algorithms are classified into two categories:

- **Interactive:** The interactive methods require multiple rounds of interaction between the parties in order for the prover to convince the verifier that the proof is valid and generally assumes the use of strong polynomials being inefficient in some real-world cases;
- **Non-interactive:** In the non-interactive methods, there is no data exchange in the attempt of showing the validity of the proof, and the prover only produces a validation certificate at the end of computation as proof of work. This type of implementation is more acceptable in the real world due to the order of polynomial-time of execution.

2.1.3 Diffie-Hellman

The rise of the Internet and its recurring innovations permitted the world to use Information Technology (IT) tools for multiple reasons and routines, from financial transactions to control our own house, just with some clicks, making our lives more flexible and dynamic. However, the dependence on technology led to a huge increase in the number of threats aimed at privacy invasion and data thefts.

Cryptography is an essential topic to keep those data safe and to prevent illegal access to private data. There are two types of cryptography, symmetric and asymmetric, and they both utilize security keys to cryptograph data, allowing the verification of the user's identity and the information itself.

Symmetric cryptography is a type of algorithm where both sides of communication use the same secret key to encrypt and decrypt the messages. Through symmetric algorithms, a message is converted to a cipher that cannot be understood by anyone who does not have the same key to decrypt it. Once this message is received by who has the same key, the algorithm decrypts it, and the original message is returned. The security of symmetric algorithms is based on how difficult it is to break a key using a brute force attack, so as bigger the key, the more trustable and safe is the algorithm.

Unlike symmetric algorithms, asymmetric cryptography or public-key cryptography operates with two keys, a public and a private. As his name suggests, the public is a key known by everyone, while the private is known and accessed just by his owner. Both keys, also known as pair of keys are used to decrypt and encrypt the messages in two situations: when Alice wants to send a message that only Bob can read and in this case, she will use the public key of Bob to encrypt the message and Bob would use his private key to decrypt, and in the situation where Alice wants to prove that she wrote a message and then she would encrypt the message with her own private key, allowing anyone to check the veracity of message with Alice's public key. Similar to symmetric algorithms, the security of asymmetric functions depends on the size of the keys.

As symmetric cryptography needs a shared secret key between the parties, another problem appeared: creating and distributing a secret key in a secure way between the parties. To solve this, Whitfield Diffie and Martin Hellman created the Diffie-Hellman key exchange algorithm in 1976. The objective of Diffie-Hellman is to allow two users to share a secret value securely in a non-secure environment, that can be used as a key for the encryption of messages (DIFFIE; HELLMAN, 2006).

Definition 2.1. ***Primitive Root*** *In modular arithmetic, a number g is a primitive root mod n, if all relatively primes of n are congruent to a power of g mod n (CORN et al., 2020).*

Diffie-Hellman is based on operations with discrete logarithms and can be described in 7 steps:

1. Initially, Alice and Bob, the parties that want to create the key, agree on two values: p, a prime number, and g, a primitive root of p. Those values can be exchanged in a non-safe network, in other words, they do not need to be kept secret from anyone;
2. Alice chooses in private a secret number a;
3. Bob do the same choosing b;
4. Alice computes A, and send it to Bob;

$$A = g^a \ mod \ p$$

5. Equally, Bob computes B and send it to Alice;

$$B = g^b \ mod \ p$$

6. Upon receiving B, Alice computes the shared key K;

$$K = B^a \ (mod \ p) = g^{ba} \ (mod \ p)$$

7. And Bob also calculates the shared K at the acquirement of A.

$$K = A^b \ (mod \ p) = g^{ab} \ (mod \ p)$$

As $g^{ab} \ (mod \ p) = g^{ba} \ (mod \ p)$, the key would always be the same and just known by the party who has the key a or b. Therefore, Alice and Bob could use the key to exchange information through a symmetric algorithm without worrying about privacy invasions or data theft. In security perspectives, if p is large enough, the computation in unfeasible, considering that a malicious person would have to calculate a from $A = g^a \ mod \ p$ or b from $B = g^b \ mod \ p$, what leads to the calculation of discrete logarithm of p that takes a considerable amount of time depending on p size.

2.1.4 Private Set Intersection

Private Set Intersection (PSI) is one of the best-studied subjects in the field of secure computation. It consists in calculate the intersection of sets from two parties without exposing any information about the items for each party (PINKAS; SCHNEIDER; ZOHNER, 2018).

An initial idea was to use a third party to handle the service. So, for example, if Alice and Bob, the parties, send their sets $S_{\mathcal{A}} = \{x_1, x_2, ..., x_n\}$ and $S_{\mathcal{B}} = \{y_1, y_2, ..., y_n\}$

to a third party Steve, he could calculate the intersection and send the result back, but in this process, he would also learn the items and Alice and Bob would lose their privacy, what violates the main principle of PSI.

A naive solution for PSI problem beyond the use of a third party would be Alice and Bob consent in use a cryptographic hash function H and calculates a new set using this function, where $S_{\mathcal{A}}^{'} = \{H(x_1), H(x_2), ..., H(x_3)\}$ and $S_{\mathcal{B}}^{'} = \{H(y_1), H(y_2), ..., H(y_3)\}$. As the same hash function is used in both sets, they could send the new set for each other and find the intersection comparing the hashes of the items. This technique works in some cases, but for sets from small spaces (phone numbers, registration codes, ...), a dishonest party could make a brute force attack and discover all the items from the other one.

Another interesting approach is use the Diffie-Hellman key exchange protocol along with PSI to achieve better results. The protocol works similarly to the naive solution. However, instead of using the same key and hash function for all items in sets, the parties would generate two sets of shared keys being one key for each item in the sets, and if a key is present on both sets, then the item is in the intersection. The limitations also include the problem when the set domain is small, but the efficiency and practicality increases as the size of sets grow (Meadows, 1986).

PSI is not important only in theory, there are several implementations for multiple use cases (CRISTOFARO; TSUDIK, 2009). For example:

- **Contact Discovery:** Check which contacts in user's address book use the same application or service;
- **Airport Security:** The airport security team could use PSI to check in a terrorist list of a foreign airline if a passenger is wanted. As neither party wills to reveal its list, they could calculate the intersection between the boarding list and terrorist lists to assure the safety of passengers;
- **Online Advertisements Performance:** Companies could compare the list of clients that received an advertisement for a specific product and the list of clients that bought that product without invading the client's privacy.

The best choice for a protocol generally depends on data size, how secure and fast the protocol should be, and if the execution platform has limited resources.

2.2 Bloom Filters

Bloom Filter is a fast and space-efficient probabilistic data structure that calculates whether an element belongs to a set or not. For the sake of efficiency, Bloom Filters can

produce false positives and, in fact, they return if the element may be in the set or definitely not.

In the original Bloom filter implementation, the methods *add*, and *contains* are available. The *add* method as his name suggests just adds an element to the set. Whilst *test* method returns *false* if the element is not in the set or *true* if the element is probably in.

The backend of a Bloom Filter is a bit vector with size m; to add some element e to a filter, a family of k cryptographic hash functions[1] is used set on k bits of the backend vector. In Figure 2 is presented a Bloom Filter containing two elements - A and B. When someone query this BloomFilter for C, a false positive occurs. Given the expected number of elements n and a tolerable probability of false positive p, the parameters m and k can be determined by:

$$m = \left\lceil \frac{n * ln(p)}{ln(\frac{1}{ln(2)^2})} \right\rceil \qquad k = \left\lceil \frac{m}{n} * ln(2) \right\rceil$$

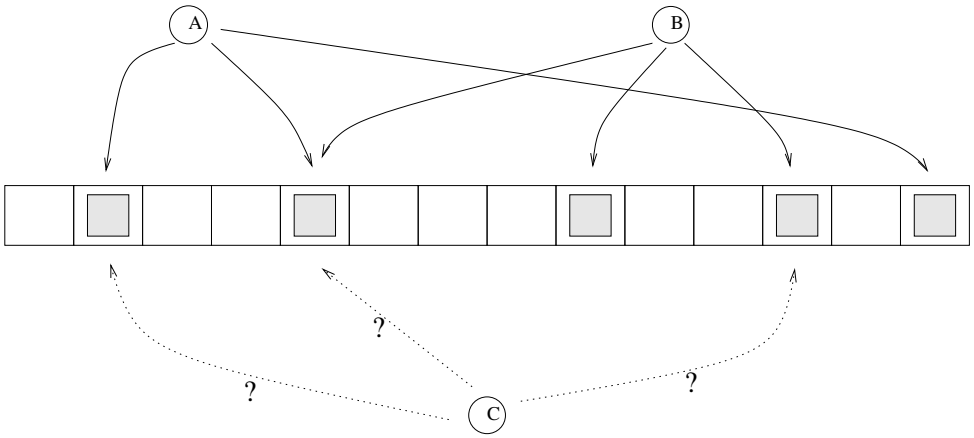


Figure 2 – A Bloom Filter with $n = 14$ and $k = 3$. The elements A and B were included in this Bloom Filter. Consulting if C is in this Bloom Filter results a false positive.

As Bloom filters are space-efficient, they have some powerful properties like:

- **Set Union:** Given two filter with the same size and using the same family of hash functions, the set union operation is given by the bitwise OR operation;
- **Set Intersection:** Similarly, the set intersection is obtained by bitwise AND;
- **Set Size:** The number of elements added to filter can be calculated approximately by this formula given X, the number of bits '1' in the filter:

[1] Cryptographic hash functions produces hash codes not suitable to be used directly as an index to an array, usually just some bits of hash code is used.

$$S = -\frac{n}{k} - \ln(1 - \frac{X}{n}).$$

Although bloom filters are built using cryptographic hash functions with properties of unidirectionality and second pre-image resistance, bloom filters offer very limited privacy in some scenarios: given a Bloom Filter determining a x that is in the filter has difficulty limited by the probability of false positives of the filter. In the context of a "dictionary attack", an opponent can easily brute force the dictionary and test each element. In some scenarios, the privacy properties are enough, and Bloom Filters are used in some cryptocurrency protocols (see (GERVAIS et al., 2014) for example).

Some authors propose advances in Bloom Filters to provide better privacy. In (LAI et al., 2006) the authors propose the partitioning of the filter in order to obtain privacy. The use of cryptographic schemes based on Pohlig-Hellman Encryption (BELLOVIN; CHESWICK, 2007), and the use of Blind Signatures and Oblivious Pseudorandom Function (NOJIMA; KADOBAYASHI, 2009) are also proposed.

2.3 Smart Contracts

The Bitcoin and Cryptocurrency revolution has brought several changes in the manner in which we comprehend and manage cash. Furthermore, those digital currencies have also helped to develop quite revolutionary technologies, with various applications in the financial world, such as the *Smart Contracts*.

Smart Contracts are self-executing digital contracts that use technology to guarantee that a contract's agreements will be fulfilled. In essence, they can be defined as programming codes described by strict rules and consequences - in the same way as a traditional document, establishing responsibilities, benefits, and penalties due to the parties in different circumstances (SZABO, 2018).

The distinction with a traditional contract is that the smart contract is public, digital, immutable, secure, and self-executing. In other words, it guarantees the security of the execution of the agreement, using Blockchain technology for this.

Thus, when a smart contract is created, and the parties had closed an agreement, the clauses are programmed, and the contract is deployed, activating the requirements automatically, simplifying the payments, and the inspection of the processes. The validation of the contract rules is done by Blockchain, which accompanies the shared data and allows direct and encrypted communication, guaranteeing more security in the whole process.

The information inserted in the agreement is automatically updated, and all the actions are executed without the risk of frauds and alterations. This is only possible

because the smart contract is immutable. However, the consequence of immutability is that even a small mistake like a typing error forces the administrator to create a new contract.

Besides, and according to Nick Szabo, some principles need to be attended for a contract be considered smart (SZABO, 2018). They are:

1. **Observability**: Ability to verify that others have fulfilled their part of the contract and to prove to others that they have fulfilled theirs;
2. **Verifiability**: Ability to prove to a third party that the contract has been executed or violated or the ability of such third parties to discover these by other manners;
3. **Privacy**: Only the responsible parties may have access to the execution of the contract.

On this wise, it is clear to see how versatile *Smart Contracts* can be, being used in elections, insurance agreements, to trade cash, properties, information, or any other items that people consider appropriate for a negotiation. Lastly, it has allowed the transformation of the conventional processes for signing contracts into an efficient, practical, and safe steps, optimizing the management and issuance of documents with suppliers or consumers of products and services.

2.4 Proof of Work (POW)

The idea came out in 1993 in an article published by Cynthia Dwork and Moni Naor (DWORK; NAOR, 1992), this scientific paper introduced a new method to reduce spam emails by using the computational power of computers. In fact, the term Proof of Work (PoW) did not appear in the article, but already had the concept of forcing the user to prove that he accomplished a task. Later in 1999, Markus Jakobsson and Ari Juels published the article "Proofs of work and bread pudding protocols" (JAKOBSSON; JUELS, 1999), which formalized the idea of the Proof of Work, a protocol where a prover demonstrates to a verifier that he has spent a certain level of time and computational effort over a specific time interval.

The protocol was not as popular until Satoshi Nakamoto released the Bitcoin *whitepaper*, given that Proof of Work was the most significant idea behind it. With this protocol, he introduced the idea of how it could be used to enable a distributed and reliable consensus.

Definition 2.2. ***Whitepaper*** *A white paper is an official document published by a government or an organization, in order to serve as a report or guide about a problem and how to face it.*

Thus, Proof of Work is an algorithm that guarantees the consensus on networks through the solution of a cryptographic problem. The protocol is expensive and takes some time to produce the necessary data, while it is easy to be verified. Bitcoin, the leading cryptocurrency on the market, uses the Proof of Work Hashcash system. Although the initial idea of Hashcash was to avoid email spammers, Satoshi applied this idea to Bitcoin transactions.

Each block in the Bitcoin blockchain has three main properties, the hash of the previous block, the list of transactions, and the nonce (Figure 3). When processing a block, the miners are forced to complete a proof of work to verify all transactions contained in it. This PoW is executed by repeatedly generating random numbers (nonces) in such a way that the hash value of the block as a whole is smaller than a certain value called target. The hardness of the proof depends directly on the total available hashing power of the network, and the target value is adjusted all the time, to guarantee that new blocks could be generated every 10 minutes. There is a very low probability of a successful generation, so it is unpredictable which miner will produce the next block first and receive the mining process's reward.

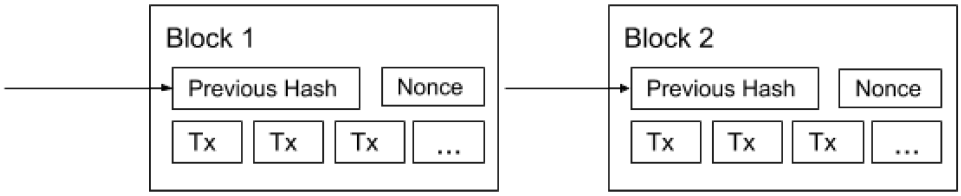


Figure 3 – The blockchain structure of Bitcoin

Considering that miners keep increasing the nonce until finding one that matches the requirement, it implies that the mining computers need to make brute force on that nonce, generating millions of hashes per second and testing them against the target, what is extremely costly and consumes a lot of time and energy. On the other hand, verifying a proof of work is simple and only requires the user to hash the block with the nonce calculated by a miner, and compare the hash value with the target.

Subsequently, when the miner finds the PoW solution, he creates a new block. This block is then transmitted to all nodes in the network, which individually checks its validity. Through this validation, the PoW ensures that the blocks cannot be added to the blockchain without performing the necessary work. Thus, a malicious node cannot easily validate blocks and add whatever it wants to the blockchain. If it tries, other network participants will discard its block, and everyone would know that the block is not valid. Even if some miners approve the invalid blocks, the mining process complicates the manipulation of blockchain state, as the malicious miner needs at least 51 % of acceptance in his invalid blocks (Leka et al., 2019).

2.5 Ethereum

The Ethereum proposal came up with Vitalik Buterin, a member of Bitcoin's own active community in 2013. The idea was to create an open-source, decentralized and distributed platform that could do more than Bitcoin, being capable of running distributed applications (DAPPS) also know as Smart Contracts based in Blockchain and use its advantages without the complexity and costs of creating a private network.

Along with Ethereum, a coin named Ether was created to be the base of the entire platform ecosystem. Beyond the normal usage of a cryptocurrency, Ether was raised to be used as payment for performing operations like the deployment of a contract or a function invocation (Leka et al., 2019) and to compensate the mining nodes for their computations.

2.5.1 Accounts

Ethereum uses Blockchain to store not only the state of your users' accounts but also source code and its associated state. Defined as objects that can store the balance of a user or the state itself of a smart contract, the accounts can be separated into two types: Externally Owned Account (EOA), accounts controlled by private keys, and Contract Account, accounts controlled by their contract code. The EOA accounts are belonged by external users who can use account's private key to send messages and sign transactions, whereas Contract accounts execute its code when it receives a message which can also deploy or interact with other contracts (SOLIDITY, 2019). The accounts own 4 fields:

1. **Nonce**: A counter to guarantee that the same transaction would not be processed multiple times;
2. **Balance**: The account's ether balance;
3. **Contract Code**: The account's contract code, if present;
4. **Storage**: The account's storage.

2.5.2 Gas

A mechanism called gas is used by Ethereum to represent the cost to pay for each computation realized by contracts. The most simple operation spends 1 unit of gas at execution, and the amount of gas needed grows in conjunction with the complexity of the contracts' source code.

On top of that, other properties were created, the gas limit, and the gas price. The purpose of the gas limit as his name suggests is to establish a limit for the use of gas avoiding denial of service attacks (DoS) as well as other spam attacks. Whereas the gas price determines the price in Ether to pay for each unit of gas spent.

In each contract deployment or function execution, the message's sender specifies the gas limit and the gas price. If the operation spends more than permitted or the sender does not have enough ether to pay the gas cost, the transaction effects are reverted, and the sender lost the gas fee spent (Leka et al., 2019).

It is essential to understand that the processing speed of transactions directly depends on how the gas price is set. Due to the fact that miners receive an amount of Ether equivalent to the total amount of gas it took them to execute a complete operation and they decide which transaction their node will process or not, they will mostly choose the transactions with the higher price, as lower prices are less lucrative for them. So, the gas price should be properly balanced, owing to a transaction with a lower price take some time or not be processed, while transactions with a higher price will be executed faster, but the Ether spent would be increasingly high (ROSIC, 2020).

2.5.3 Messages and Transactions

2.5.3.1 Transactions

Being one of the core functions in Ethereum, transactions are signed data packages that store a message to be sent from an externally owned account (EOA) forwarded by the Ethereum network and stored in the Blockchain (Leka et al., 2019). Those packages could refer to a transference of ether from an account to another, a contract creation, or a contract's method call and each transaction contain the following fields:

1. **Nonce**: A counter to guarantee that the same transaction would not be processed multiple times;
2. **Gas Price**: The fee to pay for each computation;
3. **Gas Limit**: The maximum number of gas allowed to be spent;
4. **To**: The recipient of the transaction;
5. **Value**: The amount of ether to transfer from the sender to the recipient;
6. **Data**(optional): A data field;
7. **v,r,s**: A signature that identifies the sender.

2.5.3.2 Messages

Contrarily from transactions, Messages can only be originated by Contract's Accounts, although they perform similar actions and can be used for the same objectives, ether transferences, contract creation or methods call. The main differences are that the messages are the only way how contracts can communicate with each other and they are

not stored in Blockchain, as they just exist in the running environment of Ethereum (Leka et al., 2019). The messages own 5 fields:

1. **Sender**: The sender of the message;
2. **Recipient**: The recipient of the message;
3. **Value**: The amount of ether to transfer from the sender to the recipient;
4. **Gas Limit**: The maximum number of gas allowed to be spent;
5. **Data**: A data field.

An interesting point of view about messages is that the gas price is not part of the arguments as it is in transactions. The reason for that is because the contracts do not pay gas and the gas is always paid by the EOA accounts at the creation of transactions, what does imply that the gas limit specified by a transaction is shared with all the messages that are created from its execution.

So, at the moment that a contract receives a transaction or a message, your code is executed, which can delegate a call to other methods in different contracts creating a chain of messages that will finish after the execution of them or until the gas limit is exceeded.

2.5.4 Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) is the main role in the platform ecosystem and provides a runtime environment that runs in Ethereum's network. It allows the execution of smart contracts code by compiling it to EVM bytecode, an executable code, and Contract ABI, and interface to interact with EVM bytecode. Along with those data, EVM parses the code, calculates the gas needed, executes, and mine it to store on blocks.

Once a contract is deployed, its code is compiled obtaining the EVM bytecode and the contract ABI, which is stored in the blockchain. After that, when a method is called by an external source, the call passes first by the ABI interface, what is in charge of the contract interaction and will specify the right mode to execute the method. Obviously, all participants in the Ethereum's network will execute the code concurrently on the Ethereum Virtual Machine.

Alongside EVM, Ethereum achieved more than the other blockchain systems, expanding the limits and solving more complex problems beyond the simple money transference. EVM also ensures a safe and secure running scenario as it systemically isolates the executions of codes in nodes to avoid security flaws. Indeed, if a node is compromised, then it will not affect the other nodes and the blockchain network.

EVM makes the process of creating decentralized applications (DAPPS) faster and more efficient. Instead of having to create and develop an entire blockchain for each new application, Ethereum favors thousands of different apps to be built on a single platform.

2.5.5 Blockchain and Mining

The Ethereum blockchain owns plenty of similarities with the Bitcoin blockchain. The principle distinction among them is in the blockchain architecture, in Bitcoin blockchain, the blocks are composed of the list of transactions, nonce, and a block header, while in Ethereum blockchain the list of transactions is still on the blocks, but the whole state of the network is also saved inside a block along with two other properties the block number and the difficulty (ETHEREUM, 2020).

The extra fields may appear to be costly expensive in terms of storage, but in comparison with Bitcoin is really efficient. The reason for that is because Ethereum uses a special data structure called *Patricia Tree*, a modification of Merkle Tree that stores the whole state. As the state is stored in a tree structure, the blocks only need to keep a reference for its root and the part that must be changed, saving a considered amount of stored data. Indeed, if the same strategy would be applied in Bitcoin, it could save twenty times more space.

Beyond the structure, there is also another difference: the PoW algorithm. Whilst Bitcoin implements the HashCash version, Ethereum uses *Ethash*, which produces a block every 12 seconds against 10 minutes in HashCash. The advantage is on the fastness of the transaction processing, an important topic when decentralized applications are involved (Leka et al., 2019).

3 Development

The proposal of the present work is the implementation of a fair trade protocol with item validation, where two parties that not mutually trustworthy want to negotiate the sale of information in a secure manner.

3.1 Protocol

The protocol presented below uses the concepts of Verifiable Computation, Private Set Intersection, Bloom Filter, Ethereum and Smart Contract to provide a contract that applies the rules of a fair trade. In this case, for a better explanation we would use a list of words as the item negotiated, and the situation where a party $\mathcal{A}$ wants to buy a list of strings from another party $\mathcal{B}$ using this contract. The fulfilling of this protocol is done in two parts, the preparation and the execution.

3.1.1 Preparation

As stated before, an entity $\mathcal{A}$ wants to buy a set of words from $\mathcal{B}$, which we name as $L_{\mathcal{B}}$. $\mathcal{A}$ also has his set of words $L_{\mathcal{A}}$ and in fact, he does not want all of the words from $\mathcal{B}$, but only "new words" of $L_{\mathcal{B}}$. To start the protocol, $\mathcal{A}$ and $\mathcal{B}$ agree with some conditions:

- **Bloom Filter specification:** The size of bit vector, the false positive rate and the number of hash functions that will be used to calculate the bloom filters in the next steps;
- **Collateral:** An initial collateral to be paid in case of misbehavior of the parties;
- **Timeout time:** A time limit for the parties to perform certain actions.

After that, $\mathcal{A}$ and $\mathcal{B}$ produce in private their respectives bloom filters $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$, respecting the agreed specification and using their words $L_{\mathcal{A}}$ and $L_{\mathcal{B}}$ as the items. Considering that the computation for construct the bloom filters is executed locally, there are no effective costs for it.

Consequently, $\mathcal{A}$ creates and deploys the `FairContract` in Ethereum's platform along with the established conditions, and also sends to the contract his commitment $Commit_{\mathcal{A}}$ for $BF_{\mathcal{A}}$ and the collateral previously accorded $\$_{\mathcal{A}}^{Col}$. $\mathcal{B}$ does the same next and sends his commitment $Commit_{\mathcal{B}}$ for $BF_{\mathcal{B}}$ and the collateral $\$_{\mathcal{B}}^{Col}$ (steps 1 and 2 in

Figure 4). Those commits don't trigger any action or have a special meaning, they are just a sign of interest by both parties.

Following the preparation, $\mathcal{A}$ sends $BF_{\mathcal{A}}$ and get the collateral back. This step of the protocol is symmetrical, so $\mathcal{B}$ acts in the same way (steps 3 and 4 in Figure 4). Thus, both participants are compelled to act honestly with the penalty of losing the collateral or the privacy of $L_{\mathcal{B}}$.

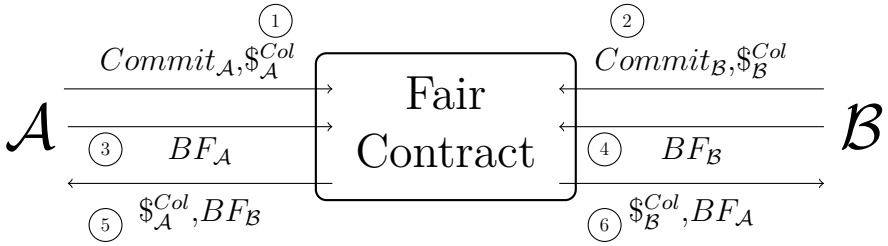


Figure 4 – `FairContract` preparation.

Knowing that the data in the blockchain is public, at the end of the preparation phase, both participants will get to know $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$ and will be able to calculate approximately the size of the set of new words

$$|L_{\mathcal{B}}| - |L_{\mathcal{A}} \cap L_{\mathcal{B}}|.$$

An important feature about using Bloom Filters in this protocol is that it allows one party to verify the honesty of the other party even before the data or values change take place. Using $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$, $\mathcal{A}$ can estimate the size of $|L_{\mathcal{A}} \cap L_{\mathcal{B}}|$, and in this way have some thoughts about the "quality" of $L_{\mathcal{B}}$: if the size of estimated intersection is too small, $\mathcal{A}$ can gives up on buying $L_{\mathcal{B}}$ concluding that the purchase of $L_{\mathcal{B}}$ isn't advantageous or that $\mathcal{B}$ created his filter with random words.

At this point, $\mathcal{B}$ creates a subset $L_{\mathcal{B}}^r$ from $L_{\mathcal{B}}$ such that the BF_r filter created from $L_{\mathcal{B}}^r$ have the following property:

$$BF_{\mathcal{A}} \vee BF_{\mathcal{B}} = BF_{\mathcal{A}} \vee BF_r,$$

where $\vee$ stands for bitwise `or` operator. In other words, $\mathcal{B}$ creates a subset of $L_{\mathcal{B}}$ containing the words that are present in $BF_{\mathcal{B}}$ and not present in $BF_{\mathcal{A}}$.

As previously said, $\mathcal{A}$ wants to buy only the "new words" of $\mathcal{B}$ and $L_{\mathcal{B}}^r$ is exactly that, the list with the words that are not present on $BF_{\mathcal{A}}$ and what $\mathcal{A}$ is really trying to purchase. $\mathcal{B}$ creates this list for two main reasons:

- In the case where $\mathcal{A}$ tries to inflate his $BF_{\mathcal{A}}$, switching some bits to "1" so that it appears to have more words, paying less for B. That doesn't happens because $\mathcal{A}$ would get no bennefits from this once it gets proportionally fewer items;

- In the case of litigation, less gas will be spent leading to lower cost (see Table 1).

Ideally, $L_{\mathcal{B}}^{r}$ should be the minimal set that meets the restriction of covering, and finding such minimum subset of coverage is NP-Hard(KARP, 1972). But, as $L_{\mathcal{B}}^{r}$ does not impact the correct execution of the protocol, only at a possible cost, a greedy algorithm is sufficient.

The initial off-chain communication to define the specifications of the protocol can be executed using traditional secure means[1]. Whereas, the bulk computations in this phase performed in the calculation of sets and bloom filters are private and each party performs the computation on your own equipment, leaving the heavy computation outside the platform of Ethereum, saving costs.

3.1.2 Execution

The execution phase starts at the moment the preparation finishes and once again, $\mathcal{A}$ and $\mathcal{B}$ uses off-chain communication to agree on some values (step 1 in Figure 5):

- $\$_{\mathcal{A}}$: the payment for $L_{\mathcal{B}}$;
- $v_{\mathcal{B}}$: a collateral to be deposited by $\mathcal{B}$ to prevent dishonesty;
- α: a value to define a penalty factor to be paid according with the situation (see Table 1).

The steps to implement the protocol are presented next, first of the honest behavior of the participants and then of the scenarios of possible misbehavior.

3.1.2.1 Honest Execution

Whenever $\mathcal{A}$ and $\mathcal{B}$ behave honestly, the protocol is straightforward:

- $\mathcal{A}$ calls the contract to start the sale, depositing $\$_{\mathcal{A}}$, the payment for the words, and sending as arguments: $v_{\mathcal{B}}$, the agreed collateral value and the α factor to calculate the penalty if it would be necessary (step 2 in Figure 5);
- $\mathcal{B}$ binds to `FairContract` sending the collateral value $\$_{\mathcal{B}}$, corresponding to $v_{\mathcal{B}}$ (step 3 in Figure 5);
- $\mathcal{B}$ sends $L_{\mathcal{B}}^{r}$ to $\mathcal{A}$ (step 4 in Figure 5). This step is also off-chain and the reason of that is because the costs of Ethereum's platform are directly related to the size of storage, and as $L_{\mathcal{B}}^{r}$ can have many items, the costs could get increasingly high

[1] Secure e-mail or instant messaging.

without the necessity, considering that an off-chain communication is enough in this case;

- Locally, $\mathcal{A}$ verifies $L^r_{\mathcal{B}}$, checking if the number of words is approximately the same as previously calculated in preparation phase and testing the words in $BF_{\mathcal{B}}$, to guarantee the veracity of the set. After the validation, if $L^r_{\mathcal{B}}$ corresponds to acceptable results, $\mathcal{A}$ sends a "sale accepted" message to contract, ending his participation on the sale (step 5 in Figure 5);
- When the contract receives the "sale accepted" message from $\mathcal{A}$, it allows $\mathcal{B}$ withdraw the payment for the list of words($\$_{\mathcal{A}}$) and the collateral once payed by him($\$_{\mathcal{B}}$), finishing the `FairContract` (step 6 in Figure 5).

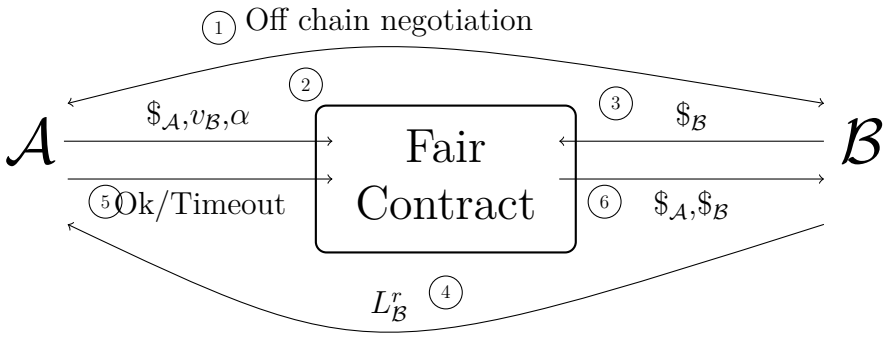


Figure 5 – `FairContract` execution for honest $\mathcal{A}$ and $\mathcal{B}$.

3.1.2.2 $\mathcal{B}$ Acting Dishonestly

As long as the parties behave honestly, the protocol is executed efficiently following the execution steps (Figure 5). But, when A or B tries to outwit the other one, an extra step is activated, the litigation, which is nothing more than contesting the result of a process, in this case, the list of the words $L^r_{\mathcal{B}}$.

In the situation where $\mathcal{B}$ is acting dishonestly and he does not send the correct $L^r_{\mathcal{B}}$ in execution phase, $\mathcal{A}$ activates the contract's *litigious* mode (Figure 6). This mode forces $\mathcal{B}$ to prove that he sent the correct list of words to $\mathcal{A}$ by sending $L^r_{\mathcal{B}}$ to the contract, which will validate $L^r_{\mathcal{B}}$ using $BF_{\mathcal{B}}$.

This mode involves two alternatives scenarios:

1. $\mathcal{B}$ sends $L^r_{\mathcal{B}}$ to `FairContract`, once $L^r_{\mathcal{B}}$ is validated using $BF_{\mathcal{B}}$, $\mathcal{B}$ receives $\$_{\mathcal{A}}$, the payment for the words, and $\alpha\$_{\mathcal{B}}$, a percentage of the collateral paid by him (Steps 1,2 and 3 in Figure 6);

2. At the activation of litigious mode, a time limit (defined in the preparation phase) starts to count, and if after it expires $\mathcal{B}$ has not sent $L_{\mathcal{B}}$ to the `FairContract`, $\mathcal{A}$ invokes `FairContract` and receives $\$_{\mathcal{A}}$ and $\$_{\mathcal{B}}$, penalizing $\mathcal{B}$ (Steps 1 and 2′ in Figure 6).

3.1.2.3 $\mathcal{A}$ Acting Dishonestly

Even, if $\mathcal{B}$ sends $L^r_{\mathcal{B}}$ correctly to $\mathcal{A}$, he could act dishonestly and calls the `FairContract` triggering the litigious mode improperly. Following the steps of the litigation, $\mathcal{B}$ will have to send $L^r_{\mathcal{B}}$ to the `FairContract`, as $L^r_{\mathcal{B}}$ will be correct, $\mathcal{A}$ will receive nothing and $\mathcal{B}$ would receive $\$_{\mathcal{A}}$ and $\alpha\$_{\mathcal{B}}$. Therefore, there is no advantage for $\mathcal{A}$ in this situation, he will have no financial benefit and will lose the confidentiality of the list just purchased.

Figure 6 – `FairContract` in litigious mode. $\mathcal{B}$ sends $L^r_{\mathcal{B}}$ to `FairContract` and receives the payment. If not, after a timeout, $\mathcal{A}$ receives the contract values.

It is important to note that the contract fails to distinguish between the scenario in which $\mathcal{A}$ acts dishonestly by calling the litigious mode improperly and the scenario in which $\mathcal{B}$ "holds" $L^r_{\mathcal{B}}$ waiting for $\mathcal{A}$ to call the litigious mode. In these two cases both participants are penalized: $\mathcal{A}$ loses $L^r_{\mathcal{B}}$ confidentiality and $\mathcal{B}$ loses (1-α)$\$_{\mathcal{B}}$. Thus, to avoid these scenarios, the values of $\$_{\mathcal{B}}$ and α need to be chosen according to an estimated value of $L^r_{\mathcal{B}}$ confidentiality. A summarization of the dishonest behaviors and their respective penalties in `FairContract` are presented at Table 1.

Dishonest Behavior	Reaction	Penalty
$\mathcal{A}$ (or $\mathcal{B}$) don't open the commit for filter	`FairContract` blocks the collateral	The collateral is lost.
$\mathcal{A}$ inflates $BF_{\mathcal{A}}$ with random bits	$\mathcal{B}$ creates $L^r_{\mathcal{B}}$ based on the difference of $BF_{\mathcal{A}}$ from $BF_{\mathcal{B}}$	$\mathcal{A}$ receives less information from $\mathcal{B}$
$\mathcal{B}$ don't send $L^r_{\mathcal{B}}$	$\mathcal{A}$ put `FairContract` in litigious mode	$\mathcal{B}$ loses $\$_{\mathcal{B}}$
$\mathcal{A}$ calls litigious improperly	$\mathcal{B}$ sends $L^r_{\mathcal{B}}$ through `FairContract`	$\mathcal{A}$ loses $L^r_{\mathcal{B}}$ confidentiality $\mathcal{B}$ loses $(1-\alpha)\$_{\mathcal{B}}$
$\mathcal{B}$ "holds" $L^r_{\mathcal{B}}$	$\mathcal{A}$ calls litigious mode, $\mathcal{B}$ sends $L^r_{\mathcal{B}}$ through `FairContract`	$\mathcal{A}$ loses $L^r_{\mathcal{B}}$ confidentiality $\mathcal{B}$ loses $(1-\alpha)\$_{\mathcal{B}}$

Table 1 – Summarization of dishonest behavior and penalties in `FairContract`.

3.2 Implementation

The fair protocol was implemented using Solidity and the Truffle Suite (Truffle & Ganache) (TRUFFLE, 2020). Truffle is the most popular Ethereum framework to create, execute, and test smart contracts. It allows developers to deploy, link libraries, write automated tests and manage network artifacts with an easy and fast API. Among their usabilities, Truffle provides the communication between the tests and the deployed contracts on Ethereum networks. Whereas, Ganache is a tool that creates a personal Ethereum blockchain where you can execute commands, run tests, inspect and debug the state of contracts easier and faster than using online Ethereum networks.

In this implementation, along with Truffle Suite, Web3.js (WEB3, 2020), a library that allows you to interact with a local or remote Ethereum node using HTTP, IPC or WebSocket, and Mocha (MOCHA, 2020), a framework for build test scenarios, helped in the creation of tests for the possible situations of contract operation.

3.2.1 Contract

The contract was implemented applying the concept of states. For each step on the phases, there is a corresponding state as an exception the *SALE_LOCKED*, which represents the end of the contract. States also have a time limit for being executed with the risk of suffering a penalty if the time expires.

```
1  enum SaleState {
2      BUYER_COMMIT,
3      SELLER_COMMIT,
4      BUYER_SEND_BLOOM_FILTER,
5      SELLER_SEND_BLOOM_FILTER,
6      BUYER_START_SALE,
7      SELLER_DEPOSIT,
8      BUYER_CONFIRM_SALE,
9      SALE_ACCEPTED,
10     LITIGIOUS_MODE,
11     SALE_LOCKED
12 }
```

Listing 3.1 – States of Contract

As stated before, the deployment of the contract is made by the buyer party ($\mathcal{A}$). During it, $\mathcal{A}$ calls the constructor sending the parameters pre-established and the seller's address starting the protocol. This step is crucial, because in this moment the timeout for the execution of functions and the addresses are stored in contract, which will be used in each next step to restrict the contract functions only to the participants.

In the preparation phase, the first two steps are the commits of the parties. The

parties send their commits by calling the *commitCollateral* function and sending along with the transaction their respective collaterals. The state is updated according to the party, so if the buyer commits, the state becomes *SELLER_COMMIT*, and if it is the seller's turn, it becomes *BUYER_SEND_BLOOM_FILTER*.

```
function commitCollateral() public payable onlyParticipant {
        if (msg.sender == buyer) {
            require(state == SaleState.BUYER_COMMIT, "Buyer cannot
                commit collateral");
            state = SaleState.SELLER_COMMIT;
            collateralBuyer = msg.value;
            endTimeState = now + timeoutDuration;
            emit Commit(msg.sender, msg.value);
        } else if (msg.sender == seller) {
            require(state == SaleState.SELLER_COMMIT, "Seller cannot
                commit collateral");
            require(now < endTimeState, "Time expired for commit the
                collateral");
            state = SaleState.BUYER_SEND_BLOOM_FILTER;
            collateralSeller = msg.value;
            endTimeState = now + timeoutDuration;
            emit Commit(msg.sender, msg.value);
        }
    }
```

Listing 3.2 – The function where both parties send their commits

Steps 3 and 4 (Figure 4) are executed by *sendBloomFilter* function. This function is similar to the *commitCollateral*, where the arguments (bloom filters) passed will be stored in the contract, the states will be updated, and the collaterals will be released to be withdrawn.

At the execution phase, the first action of the buyer is calling the *startSale* function passing the agreed parameters: penalty, factor, and the payment for the words.

```
function startSale(uint _penalty, uint _factor) public payable onlyBuyer
    {
        require(state == SaleState.BUYER_START_SALE, "Buyer cannot start
            sale");

        state = SaleState.SELLER_DEPOSIT;
        deposits[msg.sender] = msg.value;
        penalty = _penalty;
        factor = _factor;
        endTimeState = now + timeoutDuration;

        emit SaleStarted(msg.sender, msg.value, penalty, factor);
```

```
11      }
```

Listing 3.3 – The function to start the sale

After the buyer receives the words in the execution phase and if he is satisfied with the process, he calls the *acceptSale* function, where the values are relocated for the seller withdraw his collateral and the payment. However, if the words do not seem correct, he calls *refuseSale* to activate the litigious mode.

```
1  function acceptSale() external onlyBuyer {
2          require(state == SaleState.BUYER_CONFIRM_SALE, "Buyer cannot
               accept the sale");
3
4          state = SaleState.SALE_ACCEPTED;
5          uint outcome = deposits[seller].add(deposits[buyer]);
6          withdraws[seller] = withdraws[seller].add(outcome);
7
8          emit SaleAccepted(msg.sender);
9      }
```

Listing 3.4 – The function for buyer accepts the sale

```
1  function refuseSale() external onlyBuyer {
2          require(state == SaleState.BUYER_CONFIRM_SALE, "Buyer cannot
               refuse the sale");
3
4          state = SaleState.LITIGIOUS_MODE;
5          endTimeState = now + timeoutDuration;
6
7          emit SaleRefused(msg.sender);
8      }
```

Listing 3.5 – The function for buyer refuses the sale and activate the litigious mode

If the litigious mode is activated, the seller sends the words in the *sendWords* function. For the sake of efficiency, the words are passed as numbers, and the bloom filter is constructed based on that. If the *AND* bitwise operation of the bloom filter passed at the beginning of the protocol ($BF_{\mathcal{B}}$) with this new one (BF_r) results in a bloom filter equal to BF_r, so the words are correct, and the seller receives his proper money. On the other hand, if the result is different from BF_r , the buyer receives all the cash.

```
1  function sendWords(uint[] memory words) public onlySeller
       onlyIfNotExpired {
2          require(state == SaleState.LITIGIOUS_MODE, "Sale should be
               refused");
3
4          uint bloomFilter = 0;
5          for (uint i = 0; i < words.length; i++) {
6              for (uint j = 1; j <= numberOfHashes; j++) {
```

```
7                 uint256 bitPos = uint256(keccak256(abi.encodePacked(
                      words[i], j))) % 256;
8                 uint256 mask = 1 << bitPos;
9                 bloomFilter |= mask;
10            }
11        }
12
13        state = SaleState.SALE_LOCKED;
14
15        if ((bloomFilter & bloomFilterSeller) == bloomFilter) {
16            uint factorAmount = deposits[seller].mul(factor).div(100);
17            uint penaltyAmount = deposits[seller].sub(factorAmount);
18            withdraws[seller] = deposits[buyer].add(penaltyAmount);
19        } else {
20            withdraws[buyer] = deposits[buyer].add(deposits[seller]);
21        }
22
23        emit LitigiousResult(bloomFilter == bloomFilterSeller,
              bloomFilterSeller, bloomFilter);
24    }
```

Listing 3.6 – The function where seller send the words in case of litigation

The *withdraw* and *deposit* functions are just simple methods as their name suggests to withdraw the available amount for each party or deposit the collateral in the case of the seller.

The contract and his tests are available at <https://github.com/matheuscr30/Word-Sale-Fair-Contract>.

4 Results

Within the scope of this project, the proposed protocol succeeds in provides a fair trade between two parties that do not trust each other, in a secure and transparent way, without interference from third parties. However, it is necessary to consider some important factors that could become a problem when the actual scenario is taken into consideration. One of them is the cost to execute the protocol, because as mentioned before, in Ethereum's platform every computation realized by the contracts are paid in gas, which can be very expensive depending on what the contract is doing. Another factor and an essential one is the human factor. Since humans tend to always act in self-benefit, there is a need to evaluate the possible scenarios and strategies, as well as, the advantages and disadvantages of frauding the contract, so that it does not benefit the dishonesty.

Those both factors are analyzed in the next sections:

4.1 Gas Analysis

Preceding what was said about the costs of using the Ethereum's platform, the price paid for the computations is directly dependent on the gas used and its price. And even if those costs do not exist in development networks, we can use Ganache to evaluate the amount of gas and the cost in *ether* for each operation realized.

Definition 4.1. ***Gwei*** *A unit of ether that means gigawei, or better, 1,000,000,000 wei, where wei is the smallest unit of ether (1 ether =* 10^{18} *wei)*

Differently from the real environment, where for each transaction the gas price is set by the creator, Ganache asks the users to specify a gas price, that will be used for all transactions. For testing purposes, the gas price was set at 55 Gwei, which is the average price in the official Ethereum network. In other words, each unit of spent gas has cost 0.000000055 ether what is 0.00003 USD pursuant to the current dollar exchange rate on the date of this work (478.83 USD per unit of ether). Using this approach, the Table 2 presents the amount of gas used and the estimated average cost for each operation of the `FairContract` .

According the data exposed on Table 2, we can estimate the costs for the two possible scenarios of the protocol:

- **Honest Execution:** In honest executions, the buyer ($\mathcal{A}$) would approximately spend 15.24 USD to execute his operations (Deploy Contract, Commit, Send Bloom

Operation	Gas	Ether	Dollar (USD)
Deploy Contract	3002221	0.0165122	7.91
Commit	84331	0.0041322	1.98
Send Bloom Filter	75508	0.0036999	1.77
Start Sale	95643	0.03423423	2.24
Deposit	49277	0.0024146	1.16
Accept Sale	35604	0.0017446	0.83
Activate Litigious	33730	0.0016528	0.79
Send Words	59630	0.0029219	1.41
Withdraw	21921	0.0010741	0.51

Table 2 – Estimated average cost to execute the contract at Ethereum

Filter, Start Sale, Accept Sale and Withdraw). While the seller ($\mathcal{B}$) would spend 5.42 USD (Commit, Send Bloom Filter, Deposit, and Withdraw);

- **Dishonest Execution:** Now, in dishonest executions, $\mathcal{A}$ would spend 15.10 USD, being almost the same as the honest executions as he just changes the operation *Accept Sale* for *Activate Litigious*. In contrast, $\mathcal{B}$ needs to spend more gas because of the *Send Words* operation in litigious mode, costing a total of 6.83 at the end.

Thus, it is plain that $\mathcal{A}$ always pays more than $\mathcal{B}$ regardless of the behavior of participants, what is appropriate in the regard that he is the main interested in the purchase. Beyond that, if we think only on operation costs, $\mathcal{B}$ is the only party that gets prejudiced when the litigious mode is activated, and $\mathcal{A}$ could freely activate it trying to make $\mathcal{B}$ loses some money if the confidentiality of the words were not in-game. Also, it is possible to comprehend that the protocol is not optimistic for low priced purchases since 15.24 USD could be an expensive tax. For example, for a sale where the price established for the list of words is 20 USD, using the proposed protocol might not be advantageous because the buyer would have to spend almost the same price of the sale to execute the protocol. Whereas, in high-value purchases, the tax value becomes more and more insignificant with the increase in the price of the words, compensating the use of the protocol.

4.2 Game-theoretic Analysis

In this section, the protocol is analysed from the point of view of Game Theory, whither it is assumed that the participants do not behave in an honest/dishonest way but in a rational way in order to maximise the utility value of their participation, which reaches its maximal value if both parties behave honestly (SCHWARTZBACH, 2020).

The analysis of the protocol consists in build a game tree of the possible scenarios bearing in mind some properties:

- $\$_{\mathcal{A}}$: the payment cost of the words;
- $\$_{\mathcal{B}}$: the collateral paid by seller;
- g: the financial cost of the computation spent in gas to send $L^r_{\mathcal{B}}$ through contract in litigious mode;
- C: the estimate confidentiality value of $L^r_{\mathcal{B}}$;
- $per_{\mathcal{A}}$: the perceived value of $L^r_{\mathcal{B}}$, how worth are the words for $\mathcal{A}$;
- $per_{\mathcal{B}}$: the perceived value of $L^r_{\mathcal{B}}$, how worth are the words for $\mathcal{B}$.

It is important to note that the perceived values and the estimated confidentiality of $L^r_{\mathcal{B}}$ are different for each party because they have different purposes and motives for going on with the words sale. Also, we can see the perceived values like an incentive for the parties to continue the protocol, as if $per_{\mathcal{A}} > \$_{\mathcal{A}} > per_{\mathcal{B}}$, $\mathcal{A}$ would pay less than its ideal value for $L^r_{\mathcal{B}}$, and $\mathcal{B}$ would receive more that he thinks that is the ideal value of the words, being advantageous for both parties.

Overall, we can use those properties to define the utility values of $\mathcal{A}$ and $\mathcal{B}$:

- **Utility value of $\mathcal{A}$** : $U_{\mathcal{A}}$ is given by $per_{\mathcal{A}}$ - $\$_{\mathcal{A}}$, which represents the cost that $\mathcal{A}$ thinks $L^r_{\mathcal{B}}$ worth minus the actual cost payment for it;
- **Utility value of $\mathcal{B}$** : $U_{\mathcal{B}}$ is given by $\$_{\mathcal{A}}$ + $\$_{\mathcal{B}}$ - $per_{\mathcal{B}}$, which represents the payment that $\mathcal{B}$ receives from $\mathcal{A}$, plus the collateral that he sends to the contract, minus the worth value of $L^r_{\mathcal{B}}$ for him.

Considering the game tree of the protocol presented in Figure 7, we can extract some crucial knowledge about the behaviours of the parties. By commencing with the seller, if we look at the payoffs of his actions, we can see that in all cases where he sends the words, being it off-chain or through litigious mode, he gets his utility value $U_{\mathcal{B}}$ and the only difference is because he pays g when the litigation is active, proving that there is no advantageous for $\mathcal{B}$ to hold the words or do not send them off-chain in the right step. In the opposite side, if $\mathcal{B}$ does not send the words, he also will get no benefits, except when $\mathcal{A}$ accepts the sale without receiving the words, what does not happen when the parties act rationally. Besides, $\mathcal{B}$ always loses either its collateral value or the perceived value when the contract is in litigation, and he does not send the words. Indeed, $\mathcal{B}$ will

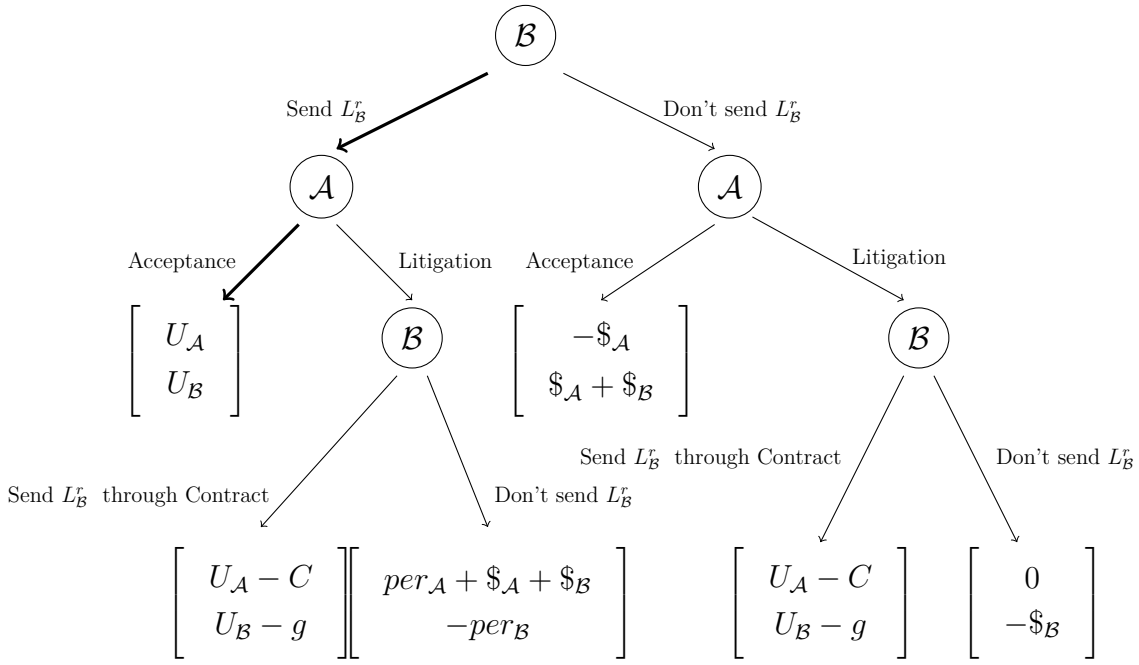


Figure 7 – Game tree of the protocol after the sale started. The first coordinate is the buyer payoff and the second is seller payoff; $\mathcal{A}$ nodes represents buyer actions and $\mathcal{B}$ nodes represents seller ones. The heavy edges denote the honest behaviour.

never benefit from dishonest actions, and therefore he will tend to act honestly to not prejudice himself.

Unlike the seller, the buyer can self-benefit from being dishonest in the situation where $\mathcal{B}$ sends $L^r_{\mathcal{B}}$ off-chain, but $\mathcal{A}$, even so, activates the litigious mode giving up on the confidentiality of $L^r_{\mathcal{B}}$. At this point, $\mathcal{B}$ will have to weigh what is worth more for him, the payment or the confidentiality of his words, in both cases $\mathcal{B}$ loses something, and to keep away from this situation, the values $\$_{\mathcal{A}}$ and $\$_{\mathcal{B}}$ need to be established considering how much the confidentiality of the words is worth to both parties. Therefore, supposing that $\$_{\mathcal{A}}$ and $\$_{\mathcal{B}}$ are balanced, whenever the litigious mode is activated, $\mathcal{A}$ will as $\mathcal{B}$ lose the confidentiality of $L^r_{\mathcal{B}}$ or the amount of the payment, which does not lead to any gain to both of them.

On this wise, it is clear to see that the litigious mode is disadvantageous for $\mathcal{A}$ and $\mathcal{B}$, and it will be executed only when necessary to avoid extra costs or loss of confidentiality when the parties act rationally. Likewise, the parties tend to be honest and reach their maximal utility value to make the execution of the protocol more efficient and cheaper.

5 Conclusion

In this work, we introduced a fair trade protocol with item validation based on Smart Contracts and Bloom Filters, where two parties that do not trust each other, want to negotiate some information. The use of Bloom Filters brought with it aspects such as efficiency and privacy, which allows items to be validated and verified at low costs without being exposed before financial transactions take place. Along with it, the Blockchain and the Smart Contract also provides reliability, security and transparency, solving some problems relevant to other online trade systems.

The proposed protocol is optimistic in the sense that if the parties behave honestly, the participation of the contract is minimal and the costs are reduced. Also, the participants are encouraged and tend to act honestly from the point of view of game theory to not suffer financial and confidentiality losses.

Even if the parties act in an honest manner, the solution still has some limitations when it comes to the cost of execution, as the price is dependent on a volatile currency and can also be expensive for certain transactions and negotiations. Albeit, the part of the protocol that requires significant CPU costs is executed privately by the parties involved, saving gas charged by Ethereum. Another limitation is in relation to Bloom Filters because if their domain set is small, it is possible to make a brute force attack and discover the items behind them.

In addition to the proposed application, the work has proved relevant as it through the analysis and results presented here, proves the logic of the contract and the trust that it brings, providing an immutable contract and a transparent way of negotiation. In future, we plan to study the viability of extending this work by using more secure alternatives to Bloom Filters.

Bibliography

ANTONOPOULOS, A.; WOOD, G. *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media, Incorporated, 2018. ISBN 9781491971949. Disponível em: <https://books.google.com.br/books?id=SedSMQAACAAJ>. Citado na página 12.

ANTONOPOULOS, A. M. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 1449374042, 9781449374044. Citado na página 11.

BASHIR, I. *Mastering Blockchain*. Packt Publishing, 2017. ISBN 9781787129290. Disponível em: <https://books.google.com.br/books?id=urkrDwAAQBAJ>. Citado na página 11.

BELLOVIN, S. M.; CHESWICK, W. R. Privacy-Enhanced Searches Using Encrypted Bloom Filters. *Columbia University Academic Commons*, p. CUCS–034–07, 2007. Citado na página 25.

BLUM, M. Coin flipping by telephone. In: *Advances in Cryptology: A Report on CRYPTO 81*. [s.n.], 1981. p. 11–15. Disponível em: </archive/crypto81/11_blum.pdf>. Citado na página 18.

BUTERIN, V. *Ethereum: A next-generation smart contract and decentralized application platform*. 2014. Accessed: 2020-08-22. Disponível em: <https://github.com/ethereum/wiki/wiki/White-Paper>. Citado na página 12.

CORN, P. et al. *Primitive Roots*. 2020. Disponível em: <https://brilliant.org/wiki/primitive-roots/>. Citado na página 21.

CRISTOFARO, E. D.; TSUDIK, G. Practical private set intersection protocols with linear computational and bandwidth complexity. *IACR Cryptology ePrint Archive*, v. 2009, p. 491, 01 2009. Citado na página 23.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. *IEEE Trans. Inf. Theor.*, IEEE Press, v. 22, n. 6, p. 644–654, set. 2006. ISSN 0018-9448. Disponível em: <https://doi.org/10.1109/TIT.1976.1055638>. Citado na página 21.

DWORK, C.; NAOR, M. Pricing via processing or combatting junk mail. In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. Berlin, Heidelberg: Springer-Verlag, 1992. (CRYPTO '92), p. 139–147. ISBN 3540573402. Citado na página 26.

ETHEREUM. *Ethereum Whitepaper*. 2020. Accessed: 2020-07-20. Disponível em: <https://ethereum.org/en/whitepaper/>. Citado na página 31.

EVANS, D.; KOLESNIKOV, V.; ROSULEK, M. A Pragmatic Introduction to Secure Multi-Party Computation. *Foundations and Trends® in Privacy and Security*, 2018. ISSN 2474-1558. Citado na página 14.

GAUR, N. et al. *Hands-On Blockchain with Hyperledger: Building Decentralized Applications with Hyperledger Fabric and Composer*. [S.l.]: Packt Publishing, 2018. ISBN 1788994523, 9781788994521. Citado na página 11.

GERVAIS, A. et al. On the privacy provisions of bloom filters in lightweight bitcoin clients. In: *Proceedings of the 30th Annual Computer Security Applications Conference*. New York, NY, USA: Association for Computing Machinery, 2014. (ACSAC '14), p. 326–335. ISBN 9781450330053. Disponível em: <https://doi.org/10.1145/2664243.2664267>. Citado na página 25.

JAKOBSSON, M.; JUELS, A. Proofs of work and bread pudding protocols. In: *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security*. NLD: Kluwer, B.V., 1999. (CMS '99), p. 258–272. ISBN 0792386000. Citado na página 26.

KARP, R. Reducibility among combinatorial problems. In: MILLER, R.; THATCHER, J. (Ed.). *Complexity of Computer Computations*. [S.l.]: Plenum Press, 1972. p. 85–103. Citado na página 34.

KOENS, T.; RAMAEKERS, C. Efficient zero-knowledge range proofs in ethereum. In: . [S.l.: s.n.], 2017. Citado na página 13.

KOSBA, A. et al. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In: *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*. [S.l.: s.n.], 2016. ISBN 9781509008247. Citado na página 16.

LAI, J. et al. Verifiable computation on outsourced encrypted data. In: KUTYŁOWSKI, M.; VAIDYA, J. (Ed.). *Computer Security - ESORICS 2014*. Cham: Springer International Publishing, 2014. p. 273–291. ISBN 978-3-319-11203-9. Citado na página 20.

LAI, P. K. Y. et al. An efficient bloom filter based solution for multiparty private matching. *Proceedings of the 2006 International Conference on Security & Management, SAM*, p. 286–292, 2006. Disponível em: <https://pdfs.semanticscholar.org/ebbd/f7b64eb8f8c8607b29c679db4b663a794d07.pdf>. Citado na página 25.

LARIMER, D. *EOS.IO White Paper*. 2017. Accessed: 2019-09-22. Disponível em: <https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>. Citado na página 12.

Leka, E. et al. Design and implementation of smart contract: A use case for geo-spatial data sharing. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.: s.n.], 2019. p. 1565–1570. Citado 5 vezes nas páginas 27, 28, 29, 30, and 31.

MATTILA, J. *The Blockchain Phenomenon – The Disruptive Potential of Distributed Consensus Architectures*. 2016. Citado na página 11.

Meadows, C. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: *1986 IEEE Symposium on Security and Privacy*. [S.l.: s.n.], 1986. p. 134–134. Citado na página 23.

MOCHA. *Mocha Project*. 2020. Accessed: 2020-05-22. Disponível em: <https://mochajs.org/>. Citado na página 37.

NAKAMOTO, S. *Bitcoin: A peer-to-peer electronic cash system*. 2008. Disponível em: <http://www.bitcoin.org/bitcoin.pdf>. Citado na página 11.

NEO. *NEO White Paper*. 2017. Accessed: 2019-09-22. Disponível em: <https://github.com/neo-project/docs/blob/master/docs/en-us/basic/whitepaper.md>. Citado na página 12.

NOJIMA, R.; KADOBAYASHI, Y. Cryptographically secure bloom-filters. *Transactions on Data Privacy*, v. 2, n. 2, p. 131–139, 2009. ISSN 18885063. Citado na página 25.

PINKAS, B.; SCHNEIDER, T.; ZOHNER, M. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, Association for Computing Machinery, New York, NY, USA, v. 21, n. 2, jan. 2018. ISSN 2471-2566. Disponível em: <https://doi.org/10.1145/3154794>. Citado na página 22.

REITWIESSNER, C. zkSNARKs in a Nutshell. *Ethereum Blog*, 2016. Citado na página 14.

ROBINSON, P. *Requirements for Ethereum Private Sidechains*. 2018. Citado na página 12.

ROSIC, A. *What is Ethereum Gas?* 2020. Citado na página 29.

SCHWARTZBACH, N. I. An incentive-compatible smart contract for decentralized commerce. *CoRR*, abs/2008.10326, 2020. Disponível em: <https://arxiv.org/abs/2008.10326>. Citado na página 42.

SOLIDITY. 2019. Accessed: 2020-07-20. Disponível em: <https://solidity.readthedocs.io/en/v0.5.11/>. Citado 2 vezes nas páginas 12 and 28.

SZABO, N. Smart contracts : Building blocks for digital markets. In: . [S.l.: s.n.], 2018. Citado 3 vezes nas páginas 12, 25, and 26.

TRUFFLE. *Truffle Suite*. 2020. Accessed: 2020-05-22. Disponível em: <https://www.trufflesuite.com>. Citado na página 37.

WEB3. *Web3 Project*. 2020. Accessed: 2020-05-22. Disponível em: <https://web3js.readthedocs.io/en/v1.2.11/>. Citado na página 37.

ZYSKIND, G.; NATHAN, O.; PENTLAND, A. *Enigma: Decentralized Computation Platform with Guaranteed Privacy*. 2015. Citado na página 15.