Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

B-Mule: A Blockchain based Secure Data Delivery Service

Relatori:
Chiar.mo Prof.
Stefano Ferretti
Chiar.mo Dot.
Mirko Zichichi

Presentata da:
Andrea Elia Tallaros

Sessione III
Anno Accademico 2021

November 24, 2021

Sommario

In questo lavoro viene presentato un servizio che consegna i messaggi sfruttando le caratteristiche native della tecnologia blockchain. L'obiettivo principale è quello di trasmettere informazioni da una parte all'altra quando una delle due parti si trova offline, senza la preoccupazione di intrusioni indesiderate. Nello specifico, l'obbiettivo é l'implementazione di un servizio che permette ad utenti che si trovano in aree non conesse di poter transmettere i propi dati ad utenti che sono conessi sulla rete. Il passaggio dei dati da una parte all'altra viene fatta da un 'mulo'. Quest'ultimo puó essere definito come l'insieme dei processi che permettono la trasmissione dei dati da un ente offline ad uno che si trova online assicurando i dati con la tecnologia blockchain. Rendendo possibile l'implementazione di questo servizio, utenti che normalmente non hanno la possibilitá di inviare messaggi o dati, possono usufruire del mulo per trasmettere dati ad un utente arbitrario mantenendosi 'scollegati' dalla rete restando peró sereni riguardo alla privacy dei dati trasmessi. Questo perché oltre ad usare la tecnologia blockchain, i dati vengono criptati con due livelli di sicurezza. Piú precisamente, é stata implementata la crittografia a chiave publica la quale rende praticamente impossibile il tampering dei dati dato che l'unico modo per decriptare il messaggio in oggetto é con la chiave privata del destinatario. Inoltre, il messaggio viene dato in pasto ad un algoritmo crittografico chiamato keccak256 il quale restituisce un hash univoco e quindi ogni cambiamento ai dati, ha come risultato un hash differente.

Caratteristiche come l'immutabilità, la sicurezza e la scalabilità sono quelle che attraggono sempre più persone, portando così ad una più ampia adozione di tale tecnologia. Ma c'è anche un lato negativo. Si discute quindi il meccanismo pricipale dietro alla architettura della tecnologia blockchain Ethereum 1.0. Fino a quando Ethereum non passerà da un protocollo PoW a un PoS, le soluzioni di secondo livello come Polygon, Optimism ecc, giocheranno un ruolo importante nell'adozione istituzionale della predetta tecnologia.

Inoltre, conoscendo le limitazioni della suddetta, si é cercato di ovviare realizzando un progetto scalabile. Nello specifico, sono stati implementati un tipo di payment channel chiamati state channel i quali servono a portare gran parte delle computazioni costose lontano dalla blockchain primaria e quindi risparmiare risorse e denaro caricando solo lo stato finale del ciclo di transazioni. Con l'implementazione dei canali, si é cercato di ridurre i costi relativi alle commissioni di transazione ed il tempo di attesa, poiché l'unica interazione con la catena è durante la dichiarazione dell'intenzione di aprire un canale e quella relativa alla chiusura.

Abbiamo dunque sfruttato le caratteristiche native della blockchain per avere un ambiente di lavoro sicuro ed in teoria a prova di manomissione. La scelta di sviluppare tutto basandosi su i concetti del web 3.0, é stata fatta per cercare di provare che l'uso di tale tencologia ha notevoli benefici quando si parla di sicurezza e di privacy dei dati - personali o non.

Abstract

Although the fact that nowadays we are continuously connected to the web may sound great, in reality it is not all as it seems regarding the architecture of data transmission. In most use cases, the data architecture is still based on the concept of an independent computer, where data is centrally stored and managed on the server and sent or retrieved by the client. Each time you communicate over the Internet, a copy of your data is sent to your service provider's server, and you lose control of your data each time. There were implemented solutions that aim to overcome this privacy issue like SSL, but even that it is applied on top of something centralized. This means that in case the central processing unit is taken down, everything will go down as well.

Moreover, there are a lot of places around the world where connection to the internet is at early stages or nonexistent meaning that people are not able to make use of all the helpful services that come with having an internet connection. Not to mention the overall cost of connecting a specific area that was previously categorized as rural.

Trying to modernize rural areas of the globe should be considered an important achievement. Nowadays being able to have an internet connection can prove life-saving. Although technological progress made sure to make the latter accessible to almost anyone, there are places that are hard to reach. The B-Mule project aims to bring messaging services to rural areas that cannot or do not have access to the internet. One of the perks of this service is the fact that an offline user can transmit data to an online one and for that, a data mule that can provide useful and trustworthy services was implemented. By doing so, people inhabiting those areas will be able to communicate with the outer world without having actual access to the net. The aim of this work is to display the feasibility of creating a data delivery service (MULE) with the aid of blockchain technology. The reason for using the aforementioned technology is that we tried to take advantage of the fact that data on the blockchain is immutable and thus, creating a delivery service that is tamper-proof would have a positive outcome in how data is delivered around the world. Using the blockchain for data transmissions is a safe way to communicate, but it comes with a cost. Trying to minimize the interactions with the online chain can assure the scalability of the application since costs can be vastly reduced. For this reason, B-Mule implements state channels in order to produce off-chain transactions maintaining costs as low as possible (nearly 0). As soon as the parties agree upon the

fact that everything prior can be transferred online, the initiator will ‘upload’ everything online essentially paying only one time instead of every time there is a task. More specifically, the application will act as a data mule that transfers messages between parties securing the latters throughout the whole transferring process with the native features of the blockchain technology like immutability, security and decentralization. On top of that, a cryptographic algorithm that will further encrypt the data set to be delivered is used. In particular, we decided to implement *keccak256* which is a one way algorithm that given some data, it will return a hash that is irreversible. This will ensure that the data that was encrypted will not be tampered because in case something like that happens, the resulting hash will be different from the first one.

Based on that, in the third chapter we will present the overall architecture of the application. Over there, it will be explained how the data passes through a two layer encryption process in order to ensure the maximum efficiency and security. Moreover, it will be described the mechanism that aims to reduce the costs and make scalable the whole application logic. Finally, the process with which the mule gets rewarded will also be introduced.

In the last chapter we will describe the respective implementation of the application as well as the results of the total costs that were needed for the completion of one full cycle meaning from the opening of the channel, the rewarding of the mule and the closure of the latter.

Contents

Chapter 1

Introduction

The internet as we know it today is a way of connecting us all seamlessly. Although this might sound like a wonderful thing, in reality, we do not control our data, nor do we have a native value settlement layer. For the majority of our use cases, data architectures are still based on the concept of stand-alone computers, where data is centrally stored and managed on a server, and sent or retrieved by a client. Every time we interact over the Internet, copies of our data get sent to the server of a service provider, and every time that happens, we lose control over our data. Even though we live in a connected world, with more and more devices getting connected to the Internet – including our watches, cars, TVs, and fridges – our data is still centrally stored: on our computers or other devices, on the USB stick, and even in the cloud. This raises issues of trust. The current Internet – with its client-server-based data infrastructure and centralized data management – has many unique points of failure, as we can see from the recurring data breaches of online service providers. It furthermore produces high costs of document handling, as well as non-transparencies along the supply chain of goods and services [1].

The Internet and the emergence of the WWW brought a data transmission protocol – TCP/IP – that made the transfer of data faster while massively reducing the transaction costs of information exchange. As years passed and the understanding of all these concepts became more clear, we saw the rise of the so-called Web2, which brought us social media and e-commerce platforms [2]. Web2 revolutionized social interactions, bringing producers and consumers of information, goods, and services closer together, and allowed us to enjoy P2P interactions on a global scale, but always with a middleman: a platform acting as a trusted intermediary between two people who do not know or trust each other. While these platforms have done a fantastic job of creating a P2P economy, with a sophisticated content discovery and value settlement layer, they also dictate all the rules of the transactions, while controlling all the data of their users.

With the adoption of the idea to decentralize the entire digital world, the concept of blockchain was introduced. In this context, blockchain seems to be a driving force of the next-generation Internet, what some refer to as the Web 3.0.

Blockchain reinvents the way data is stored and managed. It provides a unique set of data (a universal state layer) that is collectively managed. This unique state layer for the first time enables a value settlement layer for the Internet. It allows us to send files in a copy-protected way, enabling true P2P transactions without intermediaries, and it all started with the emergence of Bitcoin [1]. The Bitcoin blockchain and similar protocols are designed in a way that you would need to break into multiple houses around the globe simultaneously, which each have their own fence and alarm system, in order to breach them. This is possible but prohibitively expensive. In Web3, data is stored in each system that is part of the respective network. The management rules are formalized in the protocol and verified through the consensus of all the network participants, who are incentivized with a native network token for their activities. Blockchain, redefines the data structures in the back-end of the Web. It introduces a governance layer that runs on top of the current Internet, which allows for two people who do not know or trust each other to reach and settle agreements over the Web.

Besides that, a crucial necessity is that users, in order to interact with each other (in a centralized or decentralized way), have to be both online and as a result this creates a sort of restriction to users that do not have daily access to the web. This means that in order for the users to exchange messages, they have to stay connected otherwise they will never be able to interact with others. As a result, this work will try to overcome the issue of the necessary connection requirements in order to be able to exchange data. It's goal is pretty simple: Introduce a way with which an offline user can send information to an online one and vice versa thus minimizing the gap between people that have daily access to the web and others that struggle to obtain such a commodity. Additionally, within this work it will introduced the term of data mule which can be thought as the main protagonist of the overall architecture. The latter is the device that acquires the data while offline from a respective user, secures it with specific processes and takes care of the online delivery.

By creating such a system, it is possible to modernize areas that normally would require days or weeks to send a message from one side to another. Based on the aforementioned context, this thesis will describe the implementation of a decentralized application based on the ethereum network that will take care of the data acquisition from an offline user as well as the delivery of the respective message to the receiver. More specifically, the application will act as the data mule that transfers messages between parties securing the latters throughout the whole transferring process with the native features of the blockchain technology like immutability, security and decentralization. The application will be controlled from the browser as it has a web interface where a control panel was implemented in order to demonstrate the feasibility of the application. Moreover, an IoT device will be used in order to try to simulate a real case scenario where a sender sends the data to the MULE which will then deliver it to the receiver. For testing purposes, the receiver is not a real device, but an automated server that was implemented within the application web interface. Its job is to simulate the receiver's actions based on the data that was supposed to be delivered to the latter.

Chapter 2

State of the Art

In this chapter we will introduce the architecture of the blockchain as well as the platform Ethereum, it's characteristics and how it changed the blockchain technology.

2.1 Blockchain

A decentralized system is a peer-to-peer system in which anyone can participate. When someone joins the network, he receives a complete copy of the blockchain so practically, he synchronizes with the latter and becomes an active node in it. The blockchain's existence depends on its users. For example, if a block is created during a transaction, it will be sent to all participants of the network who will verify it. If the verification ends positively, then this block will be added to the network and all nodes will automatically have a copy of it as seen in Figure 2.4. The latter is signed by the node through a digital signature using its private key. Each user in the network has two keys.

A private key is used to create digital signatures and a public key that has two use cases:

- The first is that it acts as an address in the network and
- The second is that it is used to verify a signature or validate the identity of a sender.

The development of a blockchain is therefore aimed at enabling the exchange of digital assets between parties without trust between each other, reducing the number of intermediaries. This mechanism of data creation and management has had a strong impact across sectors as it is very efficient, automates processes reducing costs and enables the creation of new business models.

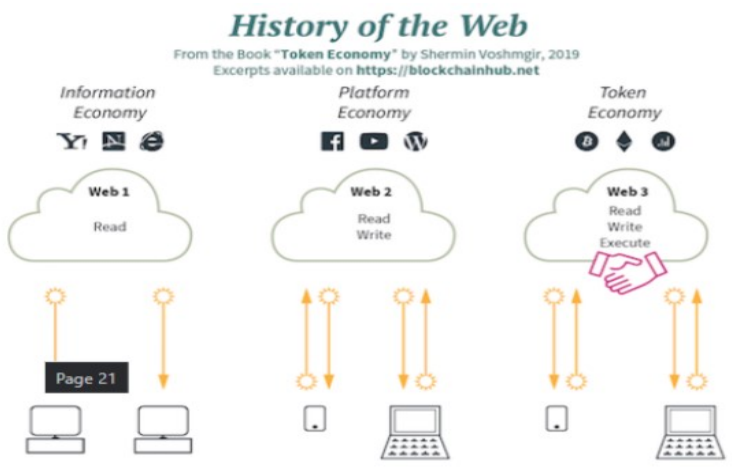


Figure 2.1: The evolution of the aforementioned architectures.

2.1.1 Traits of a blockchain

- **Immutability**: Data stored in blockchain is immutable and cannot be changed easily as explained above. Also the data is added to the block after it is approved by everyone. Those who validate the transactions and add them in the block are called miners. The validation process is called consensus algorithm and in general the most used is the one called proof of work (PoW). Proof-of-work is the underlying algorithm that sets the difficulty and the rules that miners must comply with. Mining is the "work" itself. It's the act of adding valid blocks to the chain as seen in Figure 2.3.

- **Decentralization**: A blockchain is decentralized as well as an open ledger. A ledger acts as a record holder of the transactions done and because it is visible to everyone, is called an open ledger. No individual or any organisation is in charge of the transactions. Each and every node in the blockchain network has the same exact copy of the ledger.

- **Consensus Driven**: (trust verification) each block on the blockchain is verified independently via a Consensus mechanism which provides rules for validating a block, and often uses a scarce resource (such as computing power) to show the proof that adequate effort was made. In Bitcoin, this is referred to as the mining process. This mechanism works without the use of a central authority or an explicit trust-granting agent [12].

- **Transparent**: (full transaction history) - Since the blockchain is an open source project, any party can access it and audit transactions. This creates

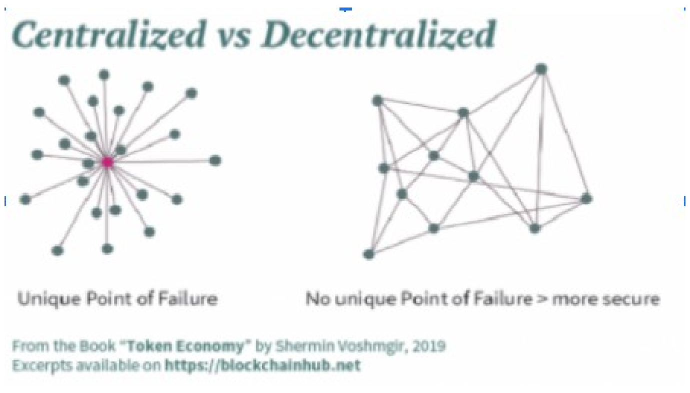

Centralized vs Decentralized
Unique Point of Failure
No unique Point of Failure > more secure
From the Book "Token Economy" by Shermin Voshmgir, 2019
Excerpts available on https://blockchainhub.net

Figure 2.2: Comparison of Centralized and Decentralized architectures.

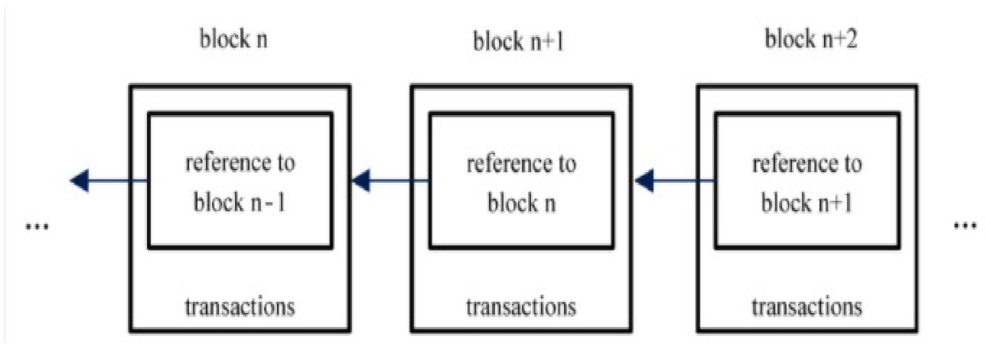


Figure 2.3: Representation of blocks inside the blockchain.

provenance under which each transaction can be tracked down to the first initiator in the genesis block.

2.2 A new Era

Ethereum was introduced by Vitalik Buterin in 2014 as an alternative to Bitcoin, optimized for the development of applications based on the blockchain [3]. The main proposed enhancement involves the introduction of the Ethereum Virtual Machine. The EVM's physical instantiation can't be described in the same way that one might point to a cloud or an ocean wave, but it does exist as one single entity maintained by thousands of connected computers running an Ethereum client. The Ethereum protocol itself exists solely for the purpose of keeping the continuous, uninterrupted, and immutable operation of this special state machine; It's the environment in which all Ethereum accounts and smart

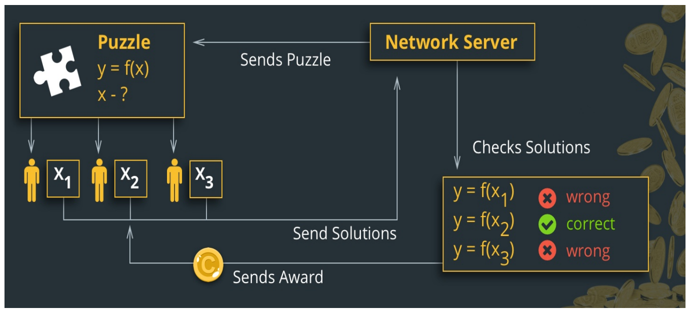


Figure 2.4: The standard PoW consensus algorithm.

contracts live. At any given block in the chain, Ethereum has one and only one 'canonical' state, and the EVM is what defines the rules for computing a new valid state from block to block [4].

The analogy of a 'distributed ledger' is often used to describe blockchains like Bitcoin, which enable a decentralized currency using fundamental tools of cryptography. A cryptocurrency behaves like a 'normal' currency because of the rules that dictate what one can and cannot do to modify the ledger. For example, a Bitcoin address cannot spend more Bitcoin than it has previously received. These rules underpin all transactions on Bitcoin and many other blockchains.

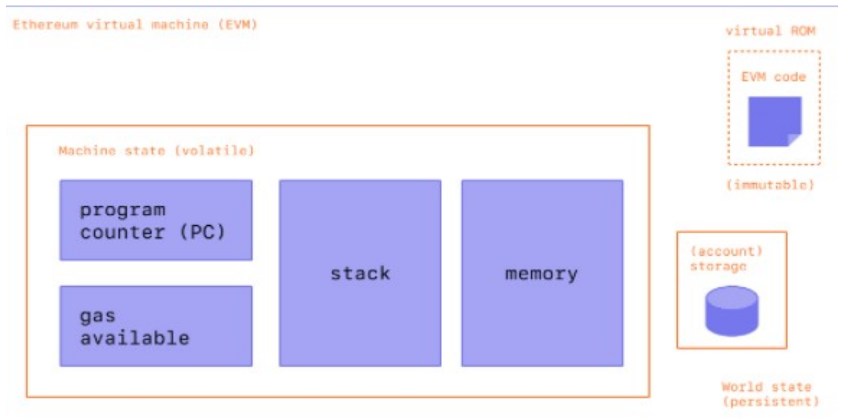


Figure 2.5: The EVM.

2.3 Smart Contracts

A smart contract is a program that runs at an address on Ethereum. They're made up of data and functions that can execute upon receiving a transaction. Here's an overview of what makes up a smart contract.
Any contract data must be assigned to a location and that means either as storage or memory. It's costly to modify storage in a smart contract so developers should consider where their data should live. Persistent data is referred to as storage and is represented by state variables. These values get stored permanently on the blockchain. Declaring the type is important in order for the contract to be able to keep track of how much storage it needs when compiling. Values that are only stored for the lifetime of a contract function's execution are called memory variables. Since these are not stored permanently on the blockchain, they are much cheaper to use.

For these more complex features, a more sophisticated analogy is required. Instead of a distributed ledger, Ethereum is a distributed state machine. Ethereum's state is a large data structure that holds not only all accounts and balances, but a machine state which can change from block to block according to a predefined set of rules, and execute arbitrary machine code. The specific rules of changing state from block to block are defined by the EVM. The latter, behaves as a mathematical function would: Given an input, it produces a deterministic output. It therefore is quite helpful to more formally describe Ethereum as having a state transition function:

$$Y(S,T) = S'$$

Given an old valid state (S) and a new set of valid transactions (T), the Ethereum state transition function Y(S, T) produces a new valid output state S'.

2.3.1 Accounts

Users on the Ethereum platform are identified by a 20-byte address, and their state.
Ethereum assumes two types of accounts:

- accounts controlled by a private key (externally owned), similar to those of Bitcoin for example, and
- accounts controlled by the code of a smart contract.

Each account is characterized by 4 fields:

- **Nonce**: in case of accounts controlled by a private key, it corresponds to the number of sent transactions, while in case of accounts referred to a smart contract it counts the number of additional contracts created by it.
- **Balance**: indicates the number of wei owned by the account. A wei indicates the smallest fraction of the respective cryptocurrency.

- **The hash of the contract code**: relevant field only for the second type of account, it represents the Keccak-256 hash of the bytecode of the smart contract.
- **Storage root**: this value is relevant only for accounts controlled by a contract: it contains the 256-bit hash of the root node of the smart contract used to represent the content of the account.

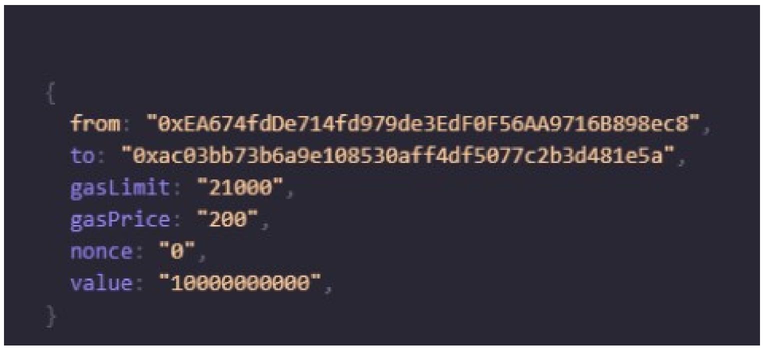
Transactions
An Ethereum transaction refers to an action initiated by an externally-owned account or,in other words, an account managed by a human, not a contract. For example, if Bob sends Alice 1 ETH, Bob's account must be debited and Alice's must be credited. This state-changing action takes place within a transaction.

Transactions, which change the state of the EVM, need to be broadcasted to the whole network. Any node can broadcast a request for a transaction to be executed on the EVM; after this happens, a miner will verify the transaction and propagate the resulting state change to the rest of the network. Transactions in Ethereum 1.0 require a fee and must be mined to become valid. A submitted transaction includes the following information:

- **Recipient** – the receiving address (if an externally-owned account, the transaction will transfer value. If it is a contract account, the transaction will execute the contract code).
- **Signature** – the identifier of the sender. This is generated when the sender's private key signs the transaction and confirms the sender has authorised this transaction.
- **Value** – amount of ETH to transfer from sender to recipient (in WEI, a denomination of ETH).
- **Data** – optional field to include arbitrary data.
- **GasLimit** – the maximum amount of gas units that can be consumed by the transaction. Units of gas represent computational steps.
- **GasPrice** – the fee the sender pays per unit of gas.

Gas is a reference to the computation required to process the transaction by a miner. Users have to pay a fee for this computation. The gasLimit and gasPrice determine the maximum transaction fee paid to the miner. For the time being, due to the fact of implementing PoW in Ethereum 1.0, the more demands there are, the higher the fees. This can be seen if we take a look at the recent spike in the cryptomarket. During the month of May 2021, Ethereum registered the highest fee for a single transaction (52.43 dollars)[5].
The transaction object will look like this:

For specific data, a signature might be required in order to have the certainty that the transaction is valid. Thus, a transaction object needs to be signed using



```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  gasPrice: "200",
  nonce: "0",
  value: "10000000000",
}
```

Figure 2.6: An Ethereum transaction summary.

the sender's private key. This proves that the transaction could only have come from the sender and was not sent fraudulently.

Transaction Lifecycle
Once the transaction has been submitted the following happens:

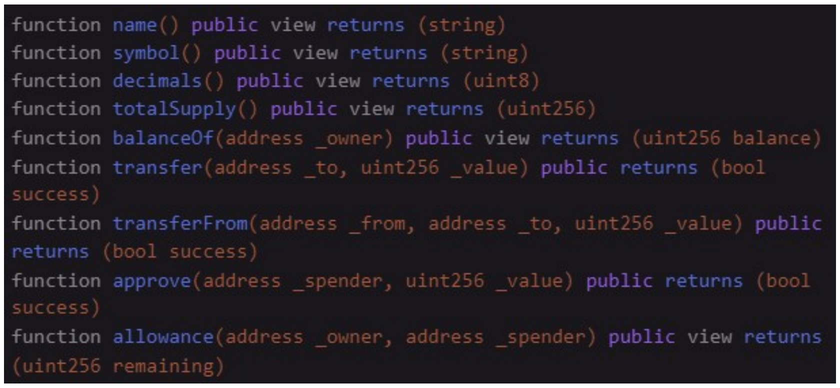
- Once you send a transaction, a cryptographic transaction hash is generated:
  0x97d99bc7729211111a21b12c933c949d4f31684f1d6954ff477d0477538ff017
- The transaction is then broadcasted to the network and included in a pool with lots of other transactions.
- A miner must pick the respective transaction, include it in a block in order to verify it and consider it "successful". Users may end up waiting at this stage if the network is busy and miners aren't able to keep up. Miners will always prioritise transactions with higher gas prices because they get to keep the fees.
- The transaction will also get a block confirmation number. This is the number of blocks created since the block that your transaction was included in. The higher the number, the greater the certainty that the transaction was processed and recognised by the network. This is because sometimes the block your transaction was included in, may not have made it into the chain. The larger the block confirmation number the more immutable the transaction is. So for higher value transactions, more block confirmations may be needed.

2.4 ERC 20 Standard

Before diving into the actual standard, it is worth getting accustomed to the term *token*. A token in the Ethereum ecosystem can be thought of as anything having a certain value, something precious. Just to illustrate some examples:

- Reputation points on an online platform.
- Skills of a character in a game of lottery tickets.
- Financial assets like a share in a company.
- Fiat currency like USD.
- Gold.

It is easily understood that there have to be specific rules that define the behaviour of such an asset in Ethereum. These particular rules are defined by the ERC (Ethereum Request for Comment) 20 Standard. This standard was proposed by Fabian Vogelsteller in November 2015 [6], and implements an API for tokens within smart contracts. It provides functionalities like transferring tokens from one account to another, getting the current token balance of an account and also the total supply of the tokens available on the network.

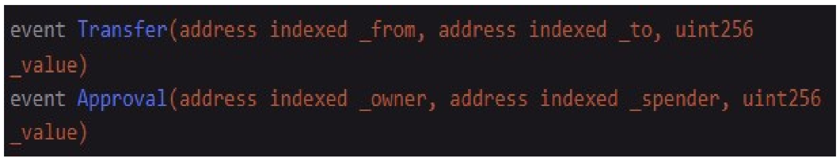


```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool
success)
function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool
success)
function allowance(address _owner, address _spender) public view returns
(uint256 remaining)
```

Figure 2.7: Interface functions that comply with the ERC 20.

Besides these, it also has some other functionalities like to approve that an amount of tokens from an account can be spent by a third party. If a smart contract implements the methods shown in Figure 2.7, once deployed, it will be responsible for keeping track of the created tokens on Ethereum.

By following the above rules any developer can create an ERC-20 token that will be used inside his application as the native currency. As already stated, Ethereum itself is not a currency. On the contrary, it is a medium that allows the creation of whole systems. Thinking of Ethereum as a baseline from which



```
event Transfer(address indexed _from, address indexed _to, uint256
_value)
event Approval(address indexed _owner, address indexed _spender, uint256
_value)
```

Figure 2.8: Event functions that comply with the ERC 20.

other ecosystems can be built is the most realistic way to start comprehending its power.
From 2014, a lot of projects are following this logic. A live example is Chainlink, a project that aims to help with an oracle network that provides reliable, tamper-proof inputs and outputs for complex smart contracts on any blockchain [7]. Another set of projects that are worth mentioning are NFTs (Non Fungible Tokens) like the one called crypto-punks. Recently, the market of NFTs, skyrocketed and the most expensive crypto-punk sold for an astronomical amount of 11 million dollars [13].

2.5 Blockchain's Limits

The main limitation is related to the transaction throughput and that is also one of the main factors that decides the rate of the adoption from institutions. This performance constraint is closely related to the inefficiency of proof-of-work consensus algorithm and the constraint that every transaction must be processed by every single node connected to the network. On the other hand, this constraint is necessary to consider a blockchain as authoritative: distributed nodes must not rely on third parties to stay updated on the status of the blockchain.

In order to scale a blockchain, increasing the block size or decreasing the block time by reducing the hash complexity is not enough. With either method, the ability to scale reaches a ceiling before it can hit the transactions necessary to compete with businesses like VISA, which handles an average of 150 million transactions every day or around 1,736 transactions per second (TPS). Right now, the Ethereum 1.0 network can only support approximately 30 transactions per second which is quite slow when compared to the amount of VISA transactions at peak hours [8].

When looking for a potential answer to the scalability problem, multiple other issues arise. For example, if the answer is only applicable for one particular blockchain, then it relies on the assumption that the particular blockchain will be the one that needs that scalability in the future; otherwise, the effort is undue or misplaced. Another consideration is to understand what the trade-offs may be. Right now, all solutions available come with limitations.

2.5.1 Scalability Trilemma

The Scalability Trilemma, a term coined by Vitalik Buterin (founder of Ethereum), refers to the tradeoffs that crypto projects must make when deciding how to optimize the underlying architecture of their own blockchain. The trilemma Vitalik is referring to involves three components:

1. **Decentralization**
2. **Scalability**
3. **Security**

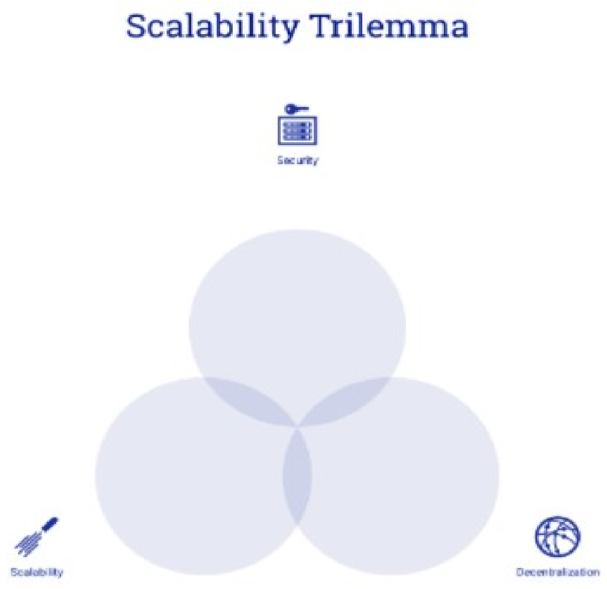


Figure 2.9: The Trilemma proposed by Vitalik.

The freedom that comes when building things with Ethereum and the fact that everything is decentralized, brings a lot of benefits to the developers and to the end-users:

- **Decentralization**, at a philosophical level, aims to bring power back to the community. By using a blockchain, where the rules of governance are literally codified and cannot be edited, one can maximize the distribution of influence, wealth and ownership across the community. The more decentralized a system is, the more secure (typically) is as well. There is no central point of failure or hack. There are, however, certain ways to hack even the most decentralized systems. Just to mention an example, one of the most common attacks is the *double-spend* attack. In this attack scenario, an attacker attempts to spend the same amount at least two times, hence *double-spend.* The attacker attempts to perform a transaction, wait for the merchant to approve it, and then revert it and spend the same currency in another transaction. In blockchains, this can be achieved by presenting a conflicting transaction possibly in a different branch.

- **Scalability**. In order for the blockchain to be used for mainstream applications such as payment systems, it needs to be able to process thousands of transactions per second. When a blockchain system is highly decentralized, scalability becomes a challenge as the ledgers on all nodes have to be updated concurrently. The less nodes you have, the more scalable the system is. It is quicker to update 10 rather than 1,000 distributed ledgers.
- **Security**. For the data on the blockchain to be trusted, the data should be protected from being leaked, lost or modified. It should be immutable and resistant to hacks (i.e. Sybil attacks, DDoS attacks, etc.). This is a basic and essential requirement.

2.6 Layer 1 Solutions

Layer 1 solutions aim to overcome the problems of Ethereum by operating directly into the core (source code) of the platform meaning that a refactoring is necessary for it to work.

2.6.1 Ethereum 2.0

Ethereum 2.0 aims to solve the aforementioned problems by creating new ways of doing things. First and most important is that the latter will no longer use a PoW protocol but rather it will start implementing PoS and this is necessary in order to combat the wastefulness and inefficiency of PoW. By doing that, consensus is achieved in a much more efficient and less intensive way. On a proof of stake blockchain, the nodes that want the chance to mine new blocks and claim the rewards can stake their crypto for a chance to become what is known as a 'validator.' This works much like a lottery: the more tickets you buy (the more you stake) the greater your chance of winning (get rewarded).

One validator is then chosen randomly to mine the new block and claim the reward, which is usually a cut of all the fees paid for the transactions contained within the block. This way of achieving consensus eliminates the need for multiple miners to use huge amounts of power in order to be allowed to mine a new block. However, Ethereum's switch to this new system is not straightforward and will be conducted in three discrete stages.

Stage 1 - The Beacon Chain
The beacon chain doesn't change anything about the Ethereum we use today. It will coordinate the network while introducing proof-of-stake to the Ethereum ecosystem as well as staking as a mechanism of reward based on your amount of ether that you expose to the public community.

Stage 2 - The merge
Eventually the current Ethereum Mainnet will "merge" with the beacon chain proof-of-stake system. This will mark the end of proof-of-work for Ethereum, and the full transition to proof-of-stake. This is planned to precede the roll out of shard chains. In the Ethereum ecosystem this process is referred as "the docking."

Stage 3 - Shard Chains
Sharding is a multi-phase upgrade to improve Ethereum's scalability and capacity. Shard chains spread the network's load across 64 new chains and they make it easier to run a node by keeping hardware requirements low. This upgrade is planned to follow the merge of the Mainnet with the Beacon Chain.

2.6.2 Sharding

Based on the trilemma mentioned before, it is impossible to achieve all three: Decentralization, Security, and Scalability simultaneously. A trade-off is necessary (you can choose any two but not all). Sharding is an attempt to solve this challenge. The latter is a proposed method of splitting Ethereum's infrastructure into smaller pieces in an attempt to scale the network.

This can be achieved by splitting the state and history of Ethereum up into partitions called "shards". For example, a sharding scheme on Ethereum might put all addresses starting with *0x00* into one shard, all addresses starting with *0x01* into another shard, etc. In the simplest form of sharding, each shard also has its own transaction history, and the effect of transactions in some shards are limited to the state of that same shard. One simple example would be a multi-asset blockchain, where there are many shards and where each one of them stores the balances and processes the transactions associated with one particular asset. In more advanced forms of sharding, it might exist some form of cross-shard communication capability, where transactions on one shard can trigger events on other shards.

For example, let's assume there is a set of validators (proof of stake nodes), who randomly get assigned the right to create shard blocks. During each slot (e.g. an 8-second period of time), for each shard in [0...999] a random validator gets selected, and given the right to create a block on a shard, which might contain up to, say, 32KB of data. Also, for each shard, a set of 100 validators get selected as attestors. The header of a block, together with at least 67 of

the attesting signatures, can be published as an object that gets included in the "main chain" (also called the beacon chain).
Note that there are now several "levels" of nodes that can exist in such a system:

- **Super-full node -** downloads the full data of the beacon chain and every shard block referenced in the beacon chain.
- **Top-level node -** processes the beacon chain blocks only, including the headers and signatures of the shard blocks, but does not download all the data of the shard blocks.
- **Single-shard node -** acts as a top-level node, but also fully downloads and verifies every collation on some specific shard that it cares more about.
- **Light node -** downloads and verifies the block headers of main chain blocks only; does not process any collation headers or transactions unless it needs to read some specific entry in the state of some specific shard, in which case it downloads the Merkle branch to the most recent collation header for that shard and from there downloads the Merkle proof of the desired value in the state.

2.7 Layer 2 Scaling Solutions

2.7.1 Optimism

Optimism is a Layer 2 scaling solution for Ethereum that can support all of Ethereum's Dapps. Instead of running all computation and data on the Ethereum network, Optimism runs computations off-chain, increasing Ethereum's transactions per second and decreasing transaction fees. Tests have shown a 143x decrease in transaction fees on the Synthetix Exchange and up to a 100x decrease on Uniswap. Since the transaction data still stays on the Ethereum network, this scaling solution does not sacrifice Ethereum's decentralisation or security for scalability.

Sequencers on Optimism are responsible for executing computations off-chain and publishing compressed transaction data onto a smart contract on Ethereum at regular checkpoints. Since computation is resource-intensive, moving it off Ethereum allows it to scale by orders of magnitude.

But what if sequencers are malicious and send fraudulent data? When the sequencer publishes the transaction data, there is a window of time where anyone can run their own computation to determine if the transaction data is fraudulent. If the data was indeed fraudulent, the verifier runs the computation on-chain and the smart contract will verify the data. The sequencer will then lose their deposit, a part of which will be sent as a reward to the verifier, and the other amount burnt. As all transactions' data is submitted on-chain, a new sequencer will be able to compute the data and replace the role of the previous sequencer. This creates:

- An economic incentive for sequencers to act honestly and,
- An economic incentive for verifiers to check on the sequencers

There are also other parties that are incentivised to verify data, such as stakeholders on Optimism or stakeholders of Dapps deployed on Optimism.

2.7.2 Side chains

Another Layer 2 scaling solution are side chains for off-chain computations. In particular the whole mechanism consists of using another blockchain, i.e. a side-chain, that runs a faster or a lighter protocol, in order to manage assets in the original blockchain, i.e. the main-chain. For instance, the Matic side-chain can ensure the asset's security using the Plasma framework and a decentralized network of Proof-of-Stake (PoS) validators. Matic strives to solve scalability and usability issues while not compromising decentralization and leveraging the existing developer community and ecosystem. The latter is a kind of para-chain scaling solution for existing platforms that provides scalability and superior user experience to DApps/user functionalities.

Key Features Highlights:

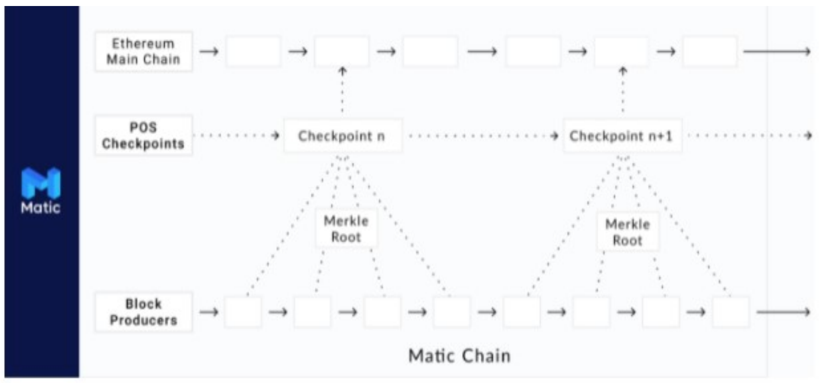


Figure 2.10: Matic Chain architecture.

- **Scalability**: Fast, low-cost and secure transactions on Matic side-chains with finality achieved on the main-chain and Ethereum as the first compatible Layer 1 base-chain
- **High Throughput**: Achieved up to 10,000 TPS on a single side-chain on internal test-net; Multiple chains to be added for horizontal scaling
- **User Experience**: Smooth UX and developer abstraction from main-chain to the Matic chain; native mobile apps and SDK with WalletConnect support
- **Security**: Matic chain operators are themselves stakers in the PoS system
- **Public Sidechains**: Matic side chains are public, permissionless and capable of supporting multiple protocols

Matic is unique both in terms of its technical approach towards Layer 2 as well as its potential support for a variety of use cases.

- Matic Layer 2 is an account-based variant of MoreVP (More Viable Plasma). The Plasma framework is used to guarantee the security of the assets on the main-chain (such as ERC-20 and ERC-721 tokens for Ethereum), while generic transactions are secured by a Proof-of-Stake network, built on top of Tendermint. Matic side-chains are essentially EVM-enabled chains and are conducive to the ready deployment of the solidity smart contracts, essentially making it an easy tool for Ethereum Developers to use for scaling their DApps/Protocols.
- Commercially, Matic side-chains are structurally effective for supporting Decentralized Finance (DeFi) protocols available in the Ethereum ecosystem.

- Matic's core philosophy is to enable DApps to compete with the user experience that is offered by centralized apps today.
- Ethereum is the first base-chain Matic Network supports, but it intends to offer support for additional base-chains, based on community suggestions and consensus, to enable an interoperable decentralized Layer 2 blockchain platform.

Matic Network solves the low transaction throughput problem by using a Block Producer layer to produce blocks a very fast rate. The system ensures decentralization using PoS checkpoints which are pushed to the Ethereum main-chain. This enables Matic to theoretically achieve 2^{16} transactions on a single side-chain.

2.7.3 State Channels

State channels are a very broad and simple way to think about blockchain interactions which could occur on the blockchain, but instead get conducted off the blockchain, without significantly increasing the risk of any participant. The most well known example of this strategy is the idea of payment channels in Bitcoin, which allow for instant fee-less payments to be sent directly between two parties [9]. In public blockchains, micropayment transactions are too expensive to be performed on-chain because the required transaction fee might be higher than the monetary value associated with the transaction. Such micropayments could be exchanged off-chain while periodically recording settlements for larger amounts on-chain. Such a solution is called a payment channel, and could also be generalised for arbitrary state updates. State channels are very similar to the concept of payment channels in Bitcoin's Lightning Network, but instead of only supporting payments, they also support general 'state updates'. By implementing the aforementioned features in our application, we can scale the system while keeping the transaction costs as low as possible. As it will be described throughout the remaining report, the goal of the application is to have only two interactions with the actual network. One for creating the intent to open a channel and one for the closure of the latter.

Using such a method, users can deposit funds into a contract and sign state updates representing moves conducted in a game of Chess without the need to communicate with the actual chain on every move (it will get costly like that). On the chain, only the final outcome of the game would be broadcasted thus marking the end of the respective game session. This allows ethereum applications to "move" expensive transactions off-chain, increasing the usefulness of the network as a whole.

State channels work by "locking up" some portion of the blockchain state into a multi-signature contract, controlled by a defined set of participants. The state that is "locked up" is called a state deposit and this might be an amount of ether or an ERC20 token. After the state deposit is locked, channel participants use off-chain messaging to exchange and sign valid ethereum transactions without deploying them into the chain. The latters are transactions that could

be put on chain anytime, but are not.
Below, a basic breakdown of the structure of a simple payment channel:

1. Part of the blockchain state is locked via multi-signature or some sort of smart contract, so that all participants must agree with each other for the state to be updated and eventually broadcasted.
2. Participants update the state amongst themselves by constructing and signing transactions that could be submitted to the blockchain, but instead are merely held onto for now. Each new update replaces previous updates.
3. Finally, participants submit the state back to the blockchain, which closes the state channel and unlocks the state again (usually in a different configuration than it started with).

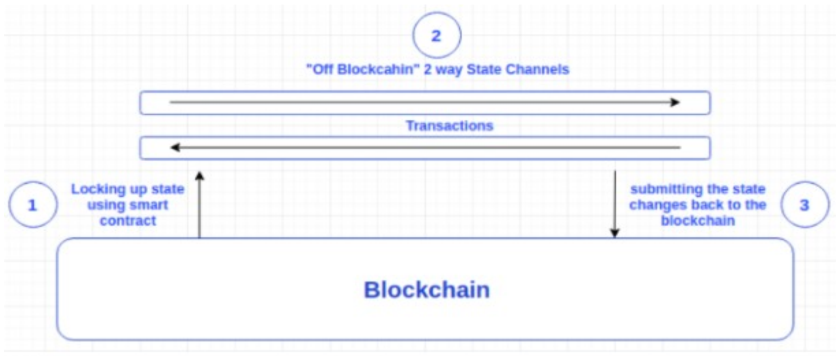

2
"Off Blockcahin" 2 way State Channels
Transactions
1
Locking up state using smart contract
submitting the state changes back to the blockchain
3
Blockchain

Figure 2.11: State Channels Architecture.

Since all exchanged transactions are equally valid as far as the blockchain is concerned, state channels need a mechanism to ensure that the latest off-chain state is the one that ultimately gets settled on the main-chain. Thus, if a party attempts to unilaterally close a channel, other parties in the channel have a period of time — a "*dispute window*" — in which they have an opportunity to submit a more recent state, thereby proving that a fraud was attempted. Once an infraction is proven, the contract handles the resolution process, which typically involves punishing the guilty party by slashing their deposited funds (though one could also simply update the system to the last valid state and proceed accordingly)[9].

If the "state" being updated between participants was a digital currency balance, then we would have a payment channel. Steps 1 and 3 (Figure 2.11), which open and close the channel, involve blockchain operations. But in step 2 an unlimited number of updates can be rapidly made without the need to involve

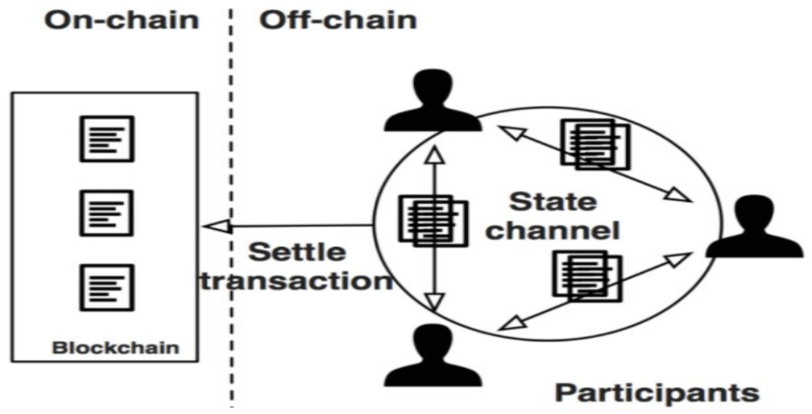


Figure 2.12: State Channels Pattern.

the blockchain at all — and this is where the power of state channels comes into play, because only steps 1 and 3 need to be published into the network, pay the fees, or wait for confirmations. In fact, with careful planning and designing, state channels can remain open almost indefinitely, and can be used as a part of a larger hub and spoke systems to power an entire economy or ecosystem.

Benefits

- **Speed** – As the blockchain is not involved in every transaction, the off-chain transactions can be settled almost instantaneously. The delay could be as low as a fraction of a second rather than whole minutes for a blockchain where we need to wait for the network to process the transaction, generate a new block with the transaction, reach consensus, and the desired number of confirmation blocks.
- **Throughput** – The number of off-chain transactions that can be processed is not limited by the blockchain's throughput, which depends on multiple factors such as the block size, the inter-block time, and the transaction fee. Thus a much higher throughput can be achieved for off-chain transactions.
- **Privacy** – Other than the settlement transaction(s), off-chain transactions do not show up in the public ledger. Thus, the detail of these intermediate off-chain transactions is not publicly visible.
- **Cost** – If a public blockchain is used, only the final settlement transaction costs are included in the blockchain. Off-chain transactions do not cost any money. In the context of payment channels network, transactions

may still pay a fee that is relatively low compared to its monitory value. For example, such networks typically charge a small percentage of the transacted amount.

Drawbacks

- **Trustworthiness** – Off-chain micropayment transactions might not be as trustworthy as on-chain transactions because they are not stored in an immutable data storage system. The intermediate state of payment channels might be lost after the payment channels are closed or do not have sufficient availability.
- **Liquidity** – To establish a payment channel, money from one or both sides of the channel participants needs to be locked up in a smart contract for the lifetime of the payment channel. The liquidity of the channel participants is thereby reduced.
- **Modifiability** – A new wallet or extension to the existing wallet is needed in order to support the micropayment protocol.

Chapter 3

B-Mule

This chapter will introduce our application called B-Mule, its goals and the overall implementation of the application. In the first part of this chapter it will be discussed the architecture starting from how the application can implement the state channels in order to safely operate and thus, the security mechanisms will also be mentioned in order to have a more complete overview of the context. Afterwards, a section dedicated to the architecture of the latter by showing the data flow diagrams for both parties (intended as sender/receiver) will follow. These diagrams' goal is to explain how a full transaction (online offline) cycle works. Finally, it ends with a section dedicated to the actual implementation by showing some snippets in order to understand how the logic of the source code works.

3.1 Goal

Trying to modernize rural areas of the globe should be considered an important achievement. Nowadays being able to have an internet connection can prove life-saving. Although the technological progress made sure to make the latter accessible to almost anyone, there are places that are hard to reach. The B-Mule project aims to bring messaging services to rural areas that cannot or do not have access to the internet. A data mule that can provide useful and trustworthy services to people inhabiting those areas in order for them to be able to communicate with the outer world without having actual access to the net.

The aim of this work is to display the feasibility of creating a data delivery service (MULE) with the aid of blockchain technology. In particular, the ideal goal would be to launch the service on the Ethereum network. The reason for using the aforementioned technology is that we try to take advantage of the fact that data on the blockchain is immutable and thus, creating a delivery service that is natively tamper proof would have a positive outcome in how data is delivered around the world. As stated in the previous chapters, using the blockchain's principles for data transmission, is a safe way to communicate but there is a cost. Trying to minimize the interactions with the chain can assure the scalability of the application since the costs can be vastly reduced. For this reason, B-Mule uses state channels in order to make off-chain transactions.

As soon as the parties agree upon the fact that everything prior can be transferred online, the initiator will 'upload' everything online essentially paying only one time instead of every time there is a task. In the image below we can see how the logic of the application works:

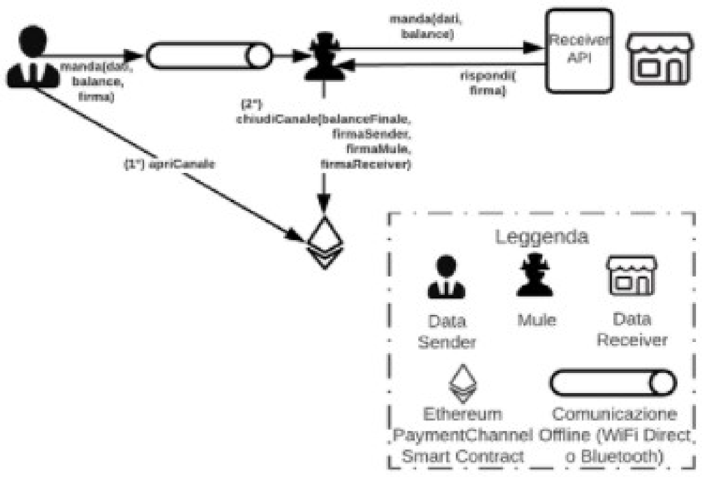


Figure 3.1: Initial Logic for the communication. (credits: Mirko Zichichi).

We are assuming that at some point in the past the user had access to the internet in order to be able to interact with the Dapp. Using state channels, the interactions with the main chain can be as low as 2. The first one happens whenever the user needs to declare the fact that he would like to open a state channel with a mule that will later deliver his message. This is mandatory since the intent to open a state channel must be recorded on-chain for obvious reasons and as such, a smart contract that will control these behaviours will be needed.

After the state channel is set up, the parties can exchange data without the need of broadcasting every step on the chain. This makes transactions almost costless and as a result, it can easily scale if necessary. The only cost related to the whole process is that of the actual payload of the message and the reward of the mule that will act as a bridge between the sender and the receiver. Whenever the state channel closes the accumulated reward will be transferred into the mule's wallet and the connection between the mule and the user can be marked as terminated and this process must be recorded online as well in order to ensure the cohesion of the state channel and the actual online chain. In case the user wants to transmit new data, he has to make sure that an open state channel exists. If not, he can open one by calling the smart contract.

3.2 Security - Encryption

3.2.1 First-layer encryption

Regarding the security of the transactions, B-mule makes use of a two layer security system. For starters, the message is encrypted using cryptographic primitives like keccak256 in order to obtain a hash. This was an implementation choice as Ethereum uses Keccak-256 in a consensus engine called Ethash. Keccak is a family of hash functions that eventually got standardized to SHA-3 (SHA256 is part of a family of hash functions called SHA-2). Ethereum called it Keccak instead of SHA-3 as it has slightly different parameters than the current SHA-3. Colloquially, Ethereum mining is never called Keccak mining because Ethash utilizes mix hashes in a DAG, which is different from the Hashcash proof-of-work.

Keccak is a family of sponge functions [11] that use, as a building block, a permutation from a set of 7 permutations. It is important to understand that it is not possible to decrypt the output of Keccak, since it isn't an encryption algorithm, but a one way hash function. Instead, it's being used for verifying that a hash value is indeed the hash output of a particular text (which is already known), simply by calculating the hash of the text, and comparing the output to the first hash value.

3.2.2 Second-layer encryption

In order to operate safely off the chain as said before, data must be encrypted and as a result we opted for an asymmetric encryption and specifically, public key encryption, or public key cryptography, which is a method of encrypting data with two different keys and making one of the keys, the public key, available for anyone to use. The other key is known as the private key. Data encrypted with the public key can only be decrypted with the private key. Implementing the encryption process following the aforementioned logic, we can assure that the only person who is eligible to decrypt the data is the private key holder. Even in the unlikely case of someone trying to access the data, the hash of the message will change, pointing out immediately the fact that something got tampered and as a result, the receiver can reject the message without the need to decrypt it first.

The increased data security provided by public key cryptography is its main benefit. Public key cryptography remains the most secure protocol (over private key cryptography) because users never need to transmit or reveal their private keys to anyone, which lessens the chances of cyber-criminals discovering an individual's secret key during the transmission.

3.3 Architecture

In this section, we will describe the architecture of our application based on what mentioned in the chapters above.

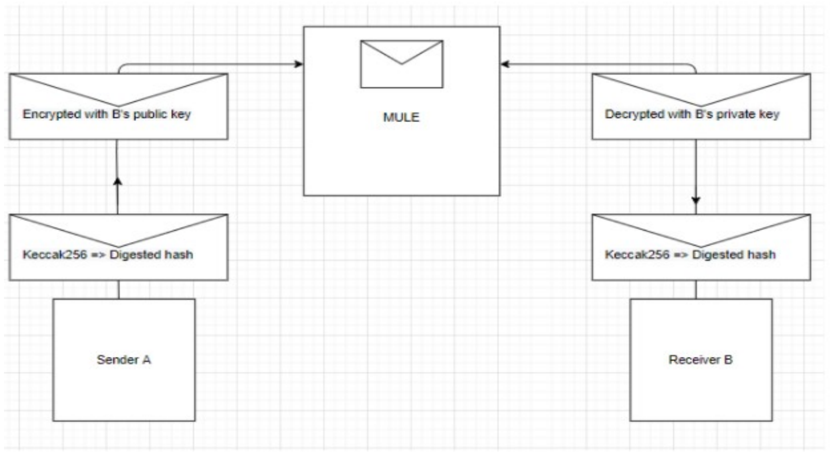

MULE
Encrypted with B's public key
Decrypted with B's private key
Keccak256 => Digested hash
Keccak256 => Digested hash
Sender A
Receiver B

Figure 3.2: Simplified view of delivering a message through a mule.

3.3.1 Front-end UI

The client is a single-page application (SPA) where all the necessary code is downloaded when the page is requested through the browser meaning that all additional resources are dynamically loaded into the application without updating or changing entirely the displayed web page.
For the realization, different tools that allow us to interact with the web 3.0 were used:

- **Metamask**: A crypto wallet that allows you to store and transact Ethereum or any other Ethereum-based (ERC- 20) tokens. You do not register it on a website, but rather install it as an extension to your Chrome or Firefox browser.
- **Ganache**: A personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.
- **Truffle**: A world-class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM), aiming to make life as a developer easier.
- **Web3.js**: A collection of libraries that allow you to interact with a local or remote ethereum node using HTTP, IPC or WebSocket.
- **React**: A Javascript library developed by Facebook to develop SPA. It is based on the concept of Components that allow to divide a user interface into reusable and independent parts

- **Eth-Crypto**: Cryptographic javascript-functions for ethereum.
- **Express.js**: A modular web framework for Node.js. It is used for the easier creation of web applications and services. Express.js simplifies development and makes it easier to write secure, modular and fast applications.

3.3.2 Smart Contracts

Since this application is coupled with the Ethereum test net, smart contracts were implemented in order to comply with the guidelines. Specifically, the main contracts created were:

- *OffChainPaymentChannel.sol*,
- *Whistle.sol*

The most important smart contract for the logic of our application is the *OffChainPaymentChannel.sol*. The latter is the handler of the state channels that users can open or close with a given Mule and as a result, whenever the latter is being called, all the following data will be registered on-chain.

Given the fact that our application is Ethereum-based, we took advantage of the ERC-20 standard mentioned in section 2.3. As stated before, following the guidelines of ERC-20, developers can create their own token that can be used inside the respective application. For our case, we minted a token called Whistle (WHL) that can be used to pay for the transactions made inside the application. For every new account created a total of 100 tokens (WHL) will be assigned to the latter. In order for the user to get the initial amount of tokens, a request to the Whistle contract has to be made. This is necessary as a link between the account address and the token contract is mandatory in order to be able to use the contract. Each time a request is made, an instance of type User is created and saved inside the contract for that specific address.

3.3.3 Application interaction

Initiating the delivery intent
The first thing that has to happen before starting to make use of the application is to connect to MetaMask with a valid account. For this purpose, we can take an account from Ganache and import it into MetaMask. Once the account has been imported we can connect to the Ethereum Testnet and start interacting with the blockchain. As stated in the previous chapters, the aim is to make the application scalable and thus we have to have the least possible number of interactions with the main chain in order to pay fewer transaction fees.

We opted for the implementation of state channels. More specifically, each user should have one and only one channel with a given Mule meaning that a client cannot have multiple open channels with the same Mule. Each channel has an identification parameter and conveniently we assign the block number that the request for opening a state channel between the user and Mule receives. As soon as a block number has been assigned to the channel, the user can start transmitting encrypted messages off-chain thus, easily scaling to whatever number wanted.

An instance of a Channel has 6 attributes.

- **The block number** and this complies with the fact that for each channel there has to be one unique identifier.
- **The Whistle** structure that has all the encrypted data as seen in Figure 3.4,
- **The PaymentInfo** structure which is an object that encapsulates the whistle,
- **A boolean** that serves as a controller for the delivery status,
- **The total cost** of that channel up until that point and
- **The timeStamp** that will be checked whenever there is a closing request

Figure 3.3 shows how the user can open a channel. It has to have a transaction with the OffChainPayment contract stored into the online chain in order to register the intent. The call, as mentioned above, will return a block number that will be used every time we want to refer to the specific channel. From now on, the user has various options:

- Create a message to be delivered,
- Close the channel

In case the latter decides to initiate a message delivery the core logic of the application will take over, create the data necessary, and in the end if everything went well, the message will be passed to the Mule already encrypted. The Mule itself is not carrying just a message, but rather a whole structure. An example of the structure of the object carried by the mule can be seen in Figure 3.4 . One important fact to note is that for every inner object like:

- A Whistle,
- a PaymentInfo or,
- a Receipt

there is always the identification parameter of the channel currently utilized (block number of the initial request). This is due to the fact that until everything is ready to be settled on the online chain, each offline transaction must be identified by an unique id.

Receiving the intent
As far it concerns the delivery, once the Mule has reached its receiver, the latter has also a couple of options:

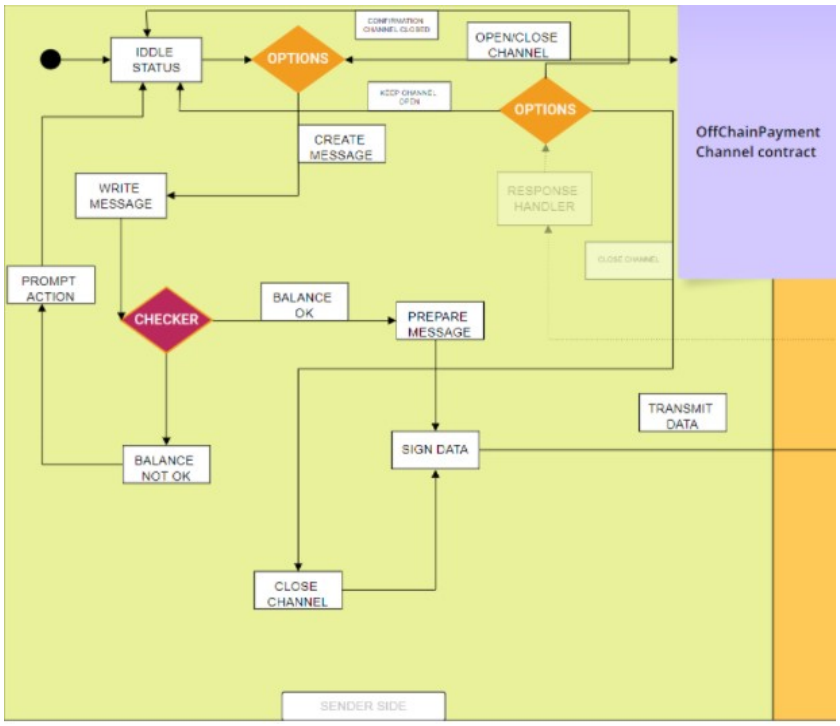


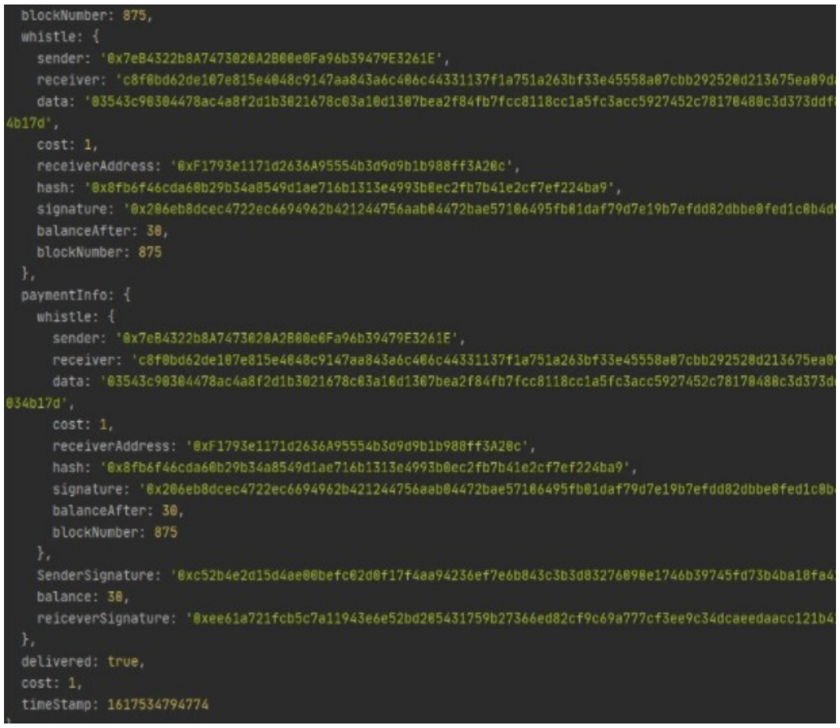
Figure 3.3: Flow chart - Sender's process.

- Initiate the decryption process, or,
- Initiate a closing request

Following the first point means that before decrypting the actual core message, the application must verify the integrity of some specific parameters like:

- The receiver address,
- If the value of the balance declared in the message intent is equal to the one received from the Mule,
- If the hash declared in the message intent is equal to the one received from the Mule

All these parameters can be retrieved following the object attributes shown in Figure 3.4. Once all the requirements have been met, the application can continue with the decryption. This process is quite straightforward and costless since the receiver does not need to encrypt and compare but rather compare the hash since keccak is a one-way encryption hash generator. If the hashes match it means that the message has its integrity and thus, can be considered safe to



```
  blockNumber: 875,
  whistle: {
    sender: '0x7eB4322b8A7473020A2B00e0Fa96b39479E3261E',
    receiver: 'c8f0bd62de107e815e4048c9147aa843a6c406c44331137f1a751a263bf33e45558a07cbb292520d213675ea89d
    data: '03543c90304478ac4a8f2d1b3021678c03a10d1307bea2f84fb7fcc8118cc1a5fc3acc5927452c78170480c3d373ddf
4b17d',
    cost: 1,
    receiverAddress: '0xF1793e1171d2636A95554b3d9d9b1b988ff3A20c',
    hash: '0x8fb6f46cda60b29b34a8549d1ae716b1313e4993b0ec2fb7b41e2cf7ef224ba9',
    signature: '0x206eb8dcec4722ec6694962b421244756aab04472bae57106495fb01daf79d7e19b7efdd82dbbe0fed1c0b4d
    balanceAfter: 30,
    blockNumber: 875
  },
  paymentInfo: {
    whistle: {
      sender: '0x7eB4322b8A7473020A2B00e0Fa96b39479E3261E',
      receiver: 'c8f0bd62de107e815e4048c9147aa843a6c406c44331137f1a751a263bf33e45558a07cbb292520d213675ea0
      data: '03543c90304478ac4a8f2d1b3021678c03a10d1307bea2f84fb7fcc8118cc1a5fc3acc5927452c78170480c3d373d
034b17d',
      cost: 1,
      receiverAddress: '0xF1793e1171d2636A95554b3d9d9b1b988ff3A20c',
      hash: '0x8fb6f46cda60b29b34a8549d1ae716b1313e4993b0ec2fb7b41e2cf7ef224ba9',
      signature: '0x206eb8dcec4722ec6694962b421244756aab04472bae57106495fb01daf79d7e19b7efdd82dbbe0fed1c0b
      balanceAfter: 30,
      blockNumber: 875
    },
    SenderSignature: '0xc52b4e2d15d4ae00befc02d0f17f4aa94236ef7e6b843c3b3d83276090e1746b39745fd73b4ba18fa4
    balance: 30,
    reiceverSignature: '0xee61a721fcb5c7a11943e6e52bd205431759b27366ed82cf9c69a777cf3ee9c34dcaeedaacc121b4
  },
  delivered: true,
  cost: 1,
  timeStamp: 1617534794774
```

Figure 3.4: Contents of an object of a channel.

open and as a result it can be signed by the receiver. Every non-intentional operation will alter the latter prompting to the fact that the tampered message will get exposed.

Signing the delivery

Once the receiver gets its message, the application requires the user to sign the transaction. Again, by using the web3.js libraries, just like the sender, the message is signed offline and thus the cycle of that message delivery intent can be marked as completed inside the object of the state channel (Figure 3.4, *@delivered:true*).

The structure of the offline signature requires some specific parameters in order to create a valid hash. These mandatory parameters are:

- **The address** of the MetaMask wallet account,
- **The block number** that has been previously assigned during the channel opening,

- **The latest balance amount** of the the respective wallet and lastly,
- **The address** of the contract that handles the channels (OfflChainPaymentChannel.sol)

For demonstration purposes, as soon as the receiver gets the message, it signs it and sends back a response that can be considered as an ‘acknowledgment’. A more in-depth look can be given by referring to Figure 3.5. On the other

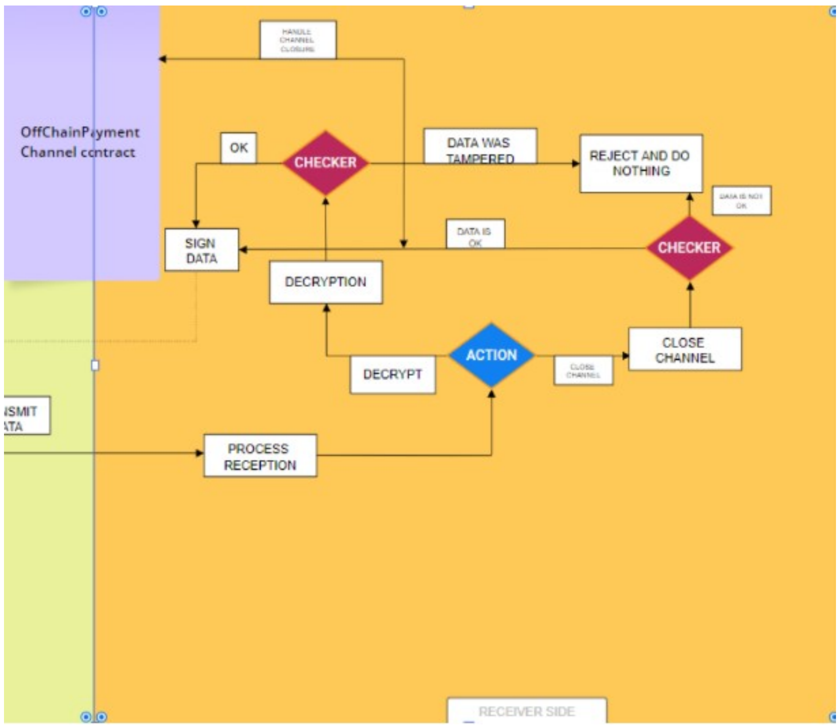
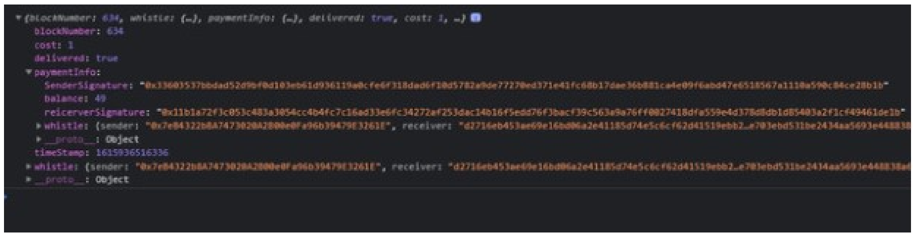


Figure 3.5: Flow chart - Receiver’s process.

hand, if the request is about closing the state channel, it means that the sender is declaring an intent to close the current channel. Before closing it and uploading the last state that both parties agreed upon on the actual blockchain, the application must have both signatures. As a result, always through MetaMask, a signature request is processed and finally the verdict is published on the blockchain. The aforementioned action is considered the second and last interaction with the main-chain.

After all the interactions, the ending structure is modeled as seen in Figure 3.6.
Due to implementation difficulties, in our application we can act as a sender.



```
▼{blockNumber: 634, whistle: {…}, paymentInfo: {…}, delivered: true, cost: 1, …}
   blockNumber: 634
   cost: 1
   delivered: true
  ▼paymentInfo:
     SenderSignature: "0x33603537bbdad52d9bf0d103eb61d936119a0cfe6f318dad6f10d5782a9de77270ed371e41fc68b17dae36b881ca4e09f6abd47e6518567a1110a590c84ce28b1b"
     balance: 49
     reicerverSignature: "0x11b1a72f3c053c483a3054cc4b4fc7c16ad33e6fc34272af253dac14b16f5edd76f3bacf39c563a9a76ff0027418dfa559e4d378d8db1d85403a2f1cf49461de1b"
    ▶whistle: {sender: "0x7e84322b8A7473020A2800e0Fa96b39479E3261E", receiver: "d2716eb453ae69e16bd06a2e41185d74e5c6cf62d41519ebb2…e703ebd531be2434aa5693e448838
    ▶__proto__: Object
   timeStamp: 1615936516336
  ▶whistle: {sender: "0x7e84322b8A7473020A2800e0Fa96b39479E3261E", receiver: "d2716eb453ae69e16bd06a2e41185d74e5c6cf62d41519ebb2…e703ebd531be2434aa5693e448838a
  ▶__proto__: Object
```

Figure 3.6: Final data structure of the concluded process.

The receiver is implemented as an ExpressJS server. One fact that is worth mentioning is that while the state channel is kept open, there is no limit to the number of off-chain transactions so the sender or the receiver can continue exchanging messages basically for free.

As far as the mule retribution, the latter will get rewarded once the channel is closed and all members involved in it agree upon the last published state. More about the retribution process can be read in the next section.

3.3.4 Mule Reward

Considering the reward of the mule, the latter will get it expressed in the native tokens of the application (WHL). As previously mentioned, a mule can get its retribution if one of the following cases are met:

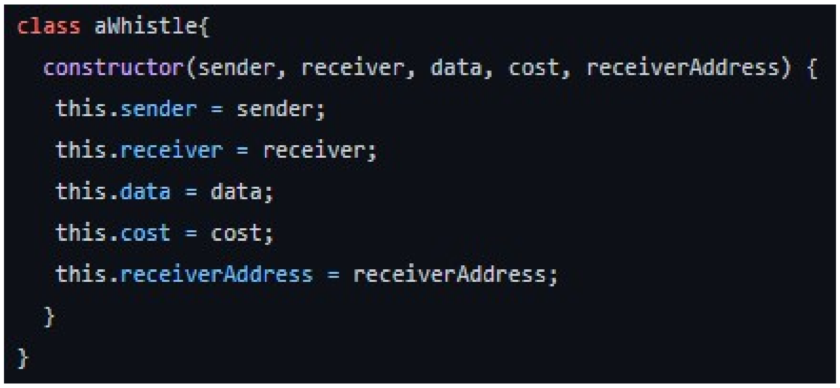
1. The user decides to close the channel or
2. The Mule itself decides to do so

One peculiarity that maybe is not expected is the fact that no deposit was made into the contract for the opening of the state channel. This means the retribution works a bit differently than expected. This is due to the fact that the application takes advantage of the ERC-20 standard function *approve(...)* that allows a third party user to pay on behalf of the allower's account up to specified amount. Not having any tokens locked up in the contract, makes it even more secure as, even in the case of an intruder, there will be no tokens to transfer to his/her account. Below we describe an example of the functions mentioned above:

Let's say that the state channel closes and the latest *timeStamp* has an aggregated cost of 5 WHL, meaning there were n transactions for a total cost of 5 WHL. Moreover, let's remember that the initial allowance (defined by *approve(...)*) was 50. Due to the fact that there was no deposit, the contract address was allowed by the client to transfer on his behalf up to 50 WHL. This implies the fact that the *OffChainPaymentChannel* contract will use the function *transferFrom(from , to, amount)* and no refunds will be made (thus less transactions in the long run). The only thing that will also need to get updated is the amount of allowance that now will be 45 WHL (unless an additional request for an allowance was made). As already mentioned, the mule can have different deliveries, but until that task is marked as completed, the mule cannot be awarded for that specific delivery. The closure request usually comes from the receiver of the data once the latter has been verified. In case something like that happens, the mule will get the reward based on the latest *timeStamp* recorded for that specific block number.

3.4 Implementation

Following what mentioned in the above section, in this one, it will be described the actual implementation of the respective concepts. A first thing that needs to be introduced is the basic structure of the object the application uses and interacts. The latter is a structure of type *aWhistle* and its implementation can be seen in the figure below:



```
class aWhistle{
  constructor(sender, receiver, data, cost, receiverAddress) {
   this.sender = sender;
   this.receiver = receiver;
   this.data = data;
   this.cost = cost;
   this.receiverAddress = receiverAddress;
  }
}
```

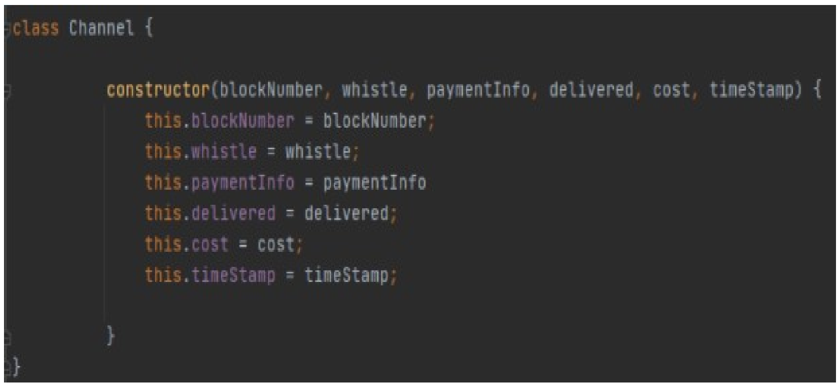
Figure 3.7: aWhistle as implemented in our Dapp.

The logic for our application is based on objects because it is easier to manipulate attributes when transferring data from and to the mule. From picture 3.7, the object *aWhistle* is a whole structure itself containing important information like:

- sender's wallet address,
- receiver's wallet address,
- the actual data to be transferred,

- the cost of the respective payload,
- receiver's public key that was used to encrypt the data

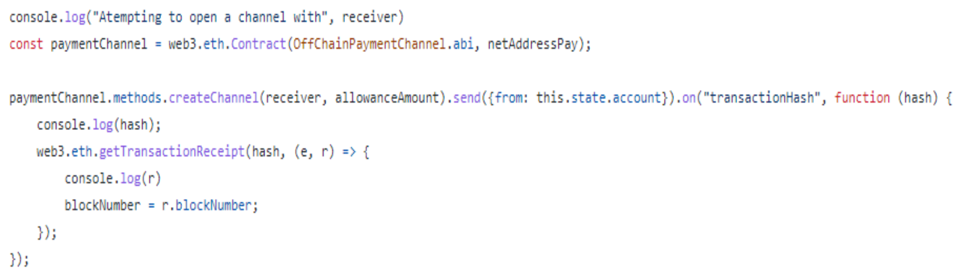
The next thing that was implemented was the medium with which these whistles will 'travel'. For convenience, the latter was implemented as an object of type *Channel* and it is implemented as follows:



```
class Channel {

        constructor(blockNumber, whistle, paymentInfo, delivered, cost, timeStamp) {
            this.blockNumber = blockNumber;
            this.whistle = whistle;
            this.paymentInfo = paymentInfo
            this.delivered = delivered;
            this.cost = cost;
            this.timeStamp = timeStamp;

        }
}
```

Figure 3.8: Channel as implemented in our Dapp.

As a result, the final structure will be an object of objects. As for the validity of the parameters, when it comes to the comparison of attributes, with an object oriented approach everything is more fluid and flexible. Once the channel has been created and the blocknumber assigned, the user can proceed in starting the exchange of messages. In figure 3.9 we can see the request to the main contract that resides on the blockchain. The resulting blocknumber that will be assigned to the Channel instance will be extracted from the transaction receipt that will be returned.

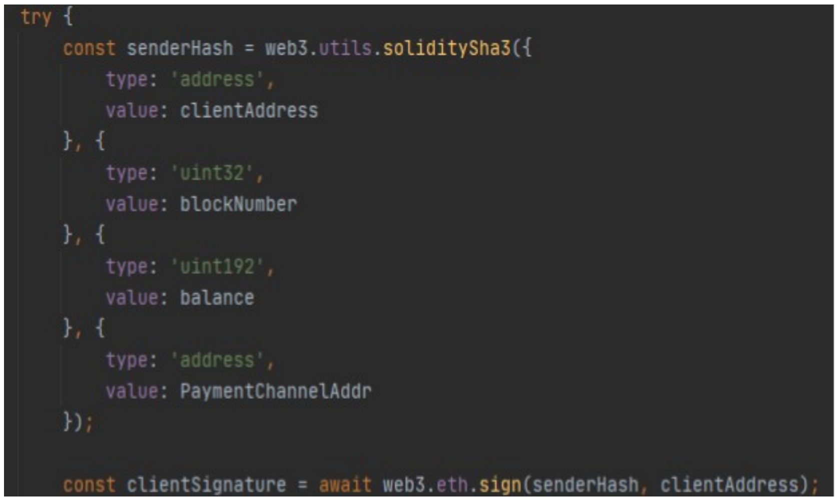


```
console.log("Atempting to open a channel with", receiver)
const paymentChannel = web3.eth.Contract(OffChainPaymentChannel.abi, netAddressPay);

paymentChannel.methods.createChannel(receiver, allowanceAmount).send({from: this.state.account}).on("transactionHash", function (hash) {
    console.log(hash);
    web3.eth.getTransactionReceipt(hash, (e, r) => {
        console.log(r)
        blockNumber = r.blockNumber;
    });
});
```

Figure 3.9: First online transaction - assigning the block number.

As mentioned earlier, messages will be transmitted offline meaning that there is a need for security. With the Web3 utils, we can sign a transaction through MetaMask and thus create a 'portable' and secured object from the device to the mule and vice versa.

In Figure 3.10 we can see how a message can be signed offline. Whenever the function *web3.eth.sign(...)* is called, MetaMask will prompt a pop up informing us for the intent of the application we are connected to. Once the transaction is registered through MetaMask, a hash is returned, which will then be added into the application native structure (aWhistle). The respective pop up can be seen in Figure 3.11.

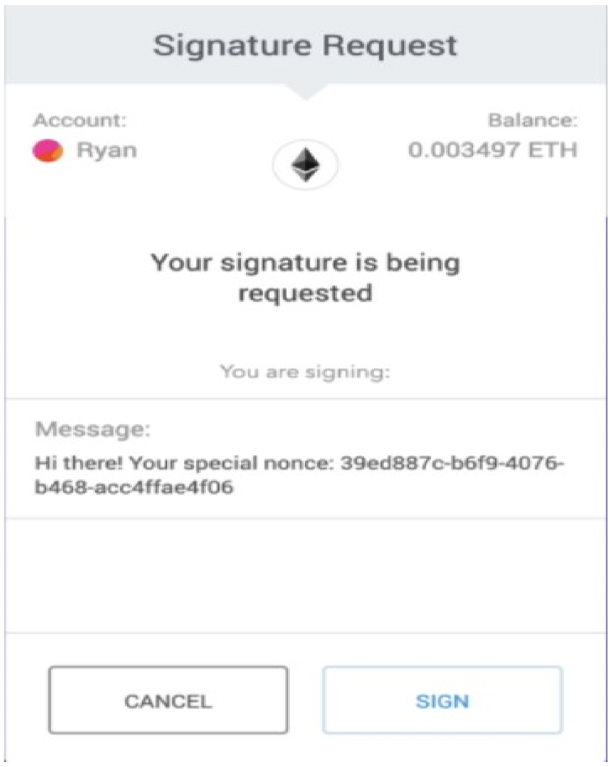


```
try {
    const senderHash = web3.utils.soliditySha3({
        type: 'address',
        value: clientAddress
    }, {
        type: 'uint32',
        value: blockNumber
    }, {
        type: 'uint192',
        value: balance
    }, {
        type: 'address',
        value: PaymentChannelAddr
    });

    const clientSignature = await web3.eth.sign(senderHash, clientAddress);
```

Figure 3.10: Web3 Utils for signing an off-chain transaction.

Encryption
As stated in the previous chapter, the encryption will be composed of two layers. The first layer is related to the encryption of the payload. The latter will get encrypted using the *keccak256* algorithm and the result of the process will be a hash that is impossible to reverse-engineer. In our application this can be achieved by using the tools provided by the EthCrypto library. As depicted in Figure 3.12, after the keccak256 encryption, we create a signature with the private key of the respective wallet as well as the payload that will later be transmitted to the mule. Finally, the public key encryption takes place (second layer encryption). For this process, the data needed is the public key of the receiver and the actual payload. The result will be stringified in order to be able to transmit it far easier by serializing it.

Decryption
Once the mule has reached the destination, it can deliver the payload. The receiver will then start the verification process by calculating by its own the hash of the payload. If the hash is the same as the one received, it means the data is



Hi there! Your special nonce: 39ed887c-b6f9-4076-b468-acc4ffae4f06

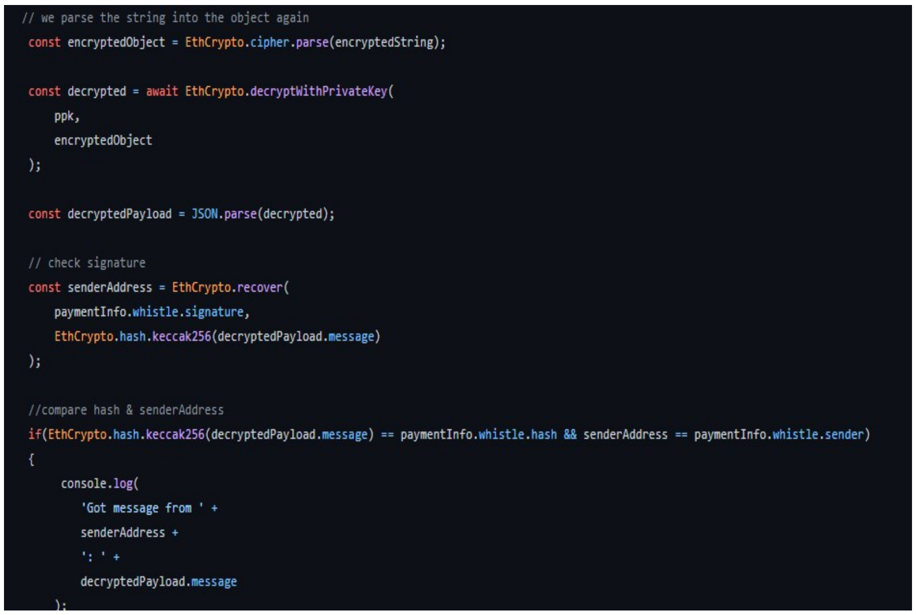
Figure 3.11: MetaMask prompt for signature.

still intact and as a result, he will proceed with the actual decryption in order to get the sent message. Specifically, two conditions have to be met in order for the receiver to sign the delivery and update the status of the respective cycle.

- The calculated hash has to be equal to the one received and,
- the sender address has to be equal to the one extracted from the *EthCrypto.revocer(...)* method.

The aforementioned logic can be seen implemented in Figure 3.13.

Closing the state channel
If the user or the mule decides to close the latter, the *OffChainPaymentChannel* contract can initiate the reward process. More specifically, as seen in Figure 3.14, by calling the respective function, the first thing that the contract does is to identify the channel, which is saved inside the contract's memory. After identifying the latter, the contract checks for the block number as it should and must be greater than 0 and if everything is valid it then proceeds with the deletion of it from the internal memory of the contract and the actual transfer



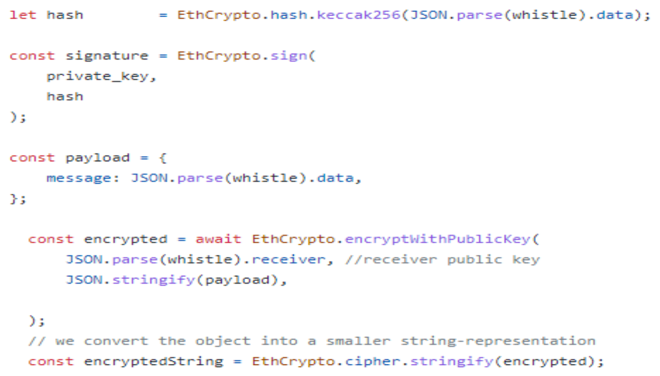
```
// we parse the string into the object again
const encryptedObject = EthCrypto.cipher.parse(encryptedString);

const decrypted = await EthCrypto.decryptWithPrivateKey(
    ppk,
    encryptedObject
);

const decryptedPayload = JSON.parse(decrypted);

// check signature
const senderAddress = EthCrypto.recover(
    paymentInfo.whistle.signature,
    EthCrypto.hash.keccak256(decryptedPayload.message)
);

//compare hash & senderAddress
if(EthCrypto.hash.keccak256(decryptedPayload.message) == paymentInfo.whistle.hash && senderAddress == paymentInfo.whistle.sender)
{
     console.log(
        'Got message from ' +
        senderAddress +
        ': ' +
        decryptedPayload.message
```

Figure 3.12: Receiver verification process.

of the funds from the sender's wallet into the mule's one. Let's remember that this is possible due to the approved amount when the channel was opened for the first time.
In the image below (Figure 3.14) we can see the snippet related to the actual retribution. The funds will go from the user's wallet into the Mule's.

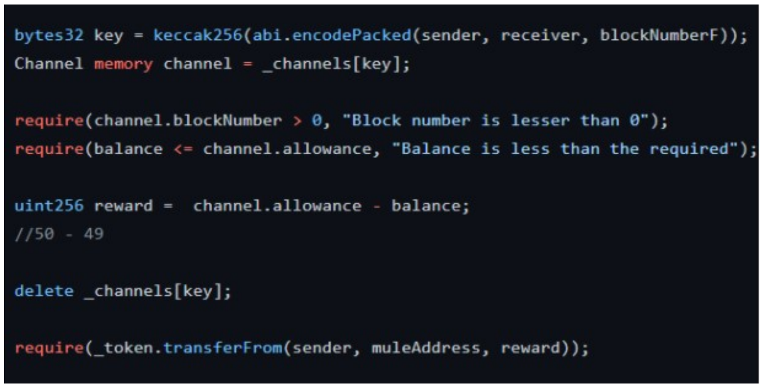
For demonstration purposes, the application will let the user initiate the intent to close an open channel with the mule. More specifically, by calling the *closeChannel(...)* function the application will interact with the smart contract *OffChainPaymentChannel*. The interaction with the aforementioned contract marks the second and last transaction with the actual blockchain.

As stated already, in this specific transaction, the final state to which all parties agree upon, will be published online. After this point the channel may stay open for further deliveries given that the updated amount of the allowance is still valid. In case there are no further funds available, the user can make a contract call and get an additional amount approved for spending otherwise the channel can safely close and as a result, for that particular blocknumber, the life cycle can be marked as completed.



```
let hash        = EthCrypto.hash.keccak256(JSON.parse(whistle).data);

const signature = EthCrypto.sign(
    private_key,
    hash
);

const payload = {
    message: JSON.parse(whistle).data,
};

  const encrypted = await EthCrypto.encryptWithPublicKey(
      JSON.parse(whistle).receiver, //receiver public key
      JSON.stringify(payload),

  );
  // we convert the object into a smaller string-representation
  const encryptedString = EthCrypto.cipher.stringify(encrypted);
```

Figure 3.13: Functions used from the EthCrypto library.



```
bytes32 key = keccak256(abi.encodePacked(sender, receiver, blockNumberF));
Channel memory channel = _channels[key];

require(channel.blockNumber > 0, "Block number is lesser than 0");
require(balance <= channel.allowance, "Balance is less than the required");

uint256 reward =  channel.allowance - balance;
//50 - 49

delete _channels[key];

require(_token.transferFrom(sender, muleAddress, reward));
```

Figure 3.14: Closing the state Channel rewarding the mule.

Chapter 4

Demo - Results

In this chapter, the key components for the realization of the demonstration will be described. We will try to simulate the data acquisition process with the aid of an IoT device (ESP32). The device is thought of as the Sender, and the localhost as the data-mule/receiver. For testing purposes, the idea is to make them exchange information through a WiFi connection. Below the device that will act as the user that sends a message.

4.1 Hardware - Software

In Figure 4.1, we can see an az-delivery ESP32 prototyping module and the power connector which is a simple USB (Figure 4.1.1).

The module was programmed into the Arduino IDE and as a result, the code was then flashed inside its memory.

In order to allow communication from the module to the localhost, we created a static IP with the help of XAMPP in which PHP scripts that captured the data from the IoT module were written. These scripts processed the latter and sent it over to the application. The application itself is mainly built with React, thus nodeJS had also an important role in the initial setup.

Figure 4.1 ESP32

Figure 4.1.1 USB

4.2 Demo Implementation

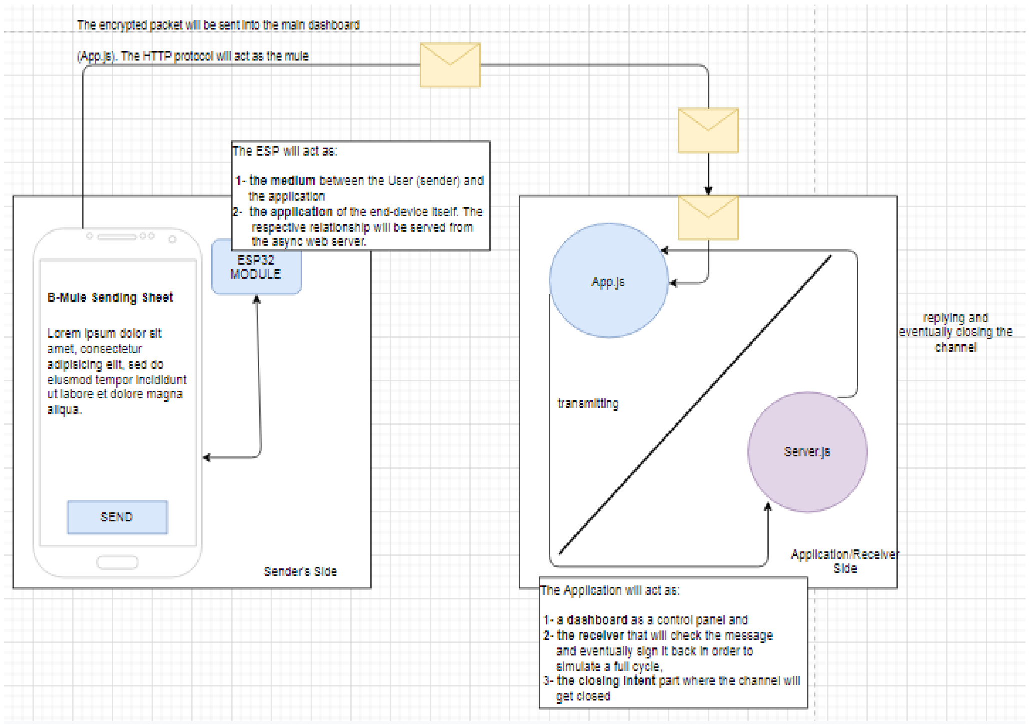

The encrypted packet will be sent into the main dashboard
(App.js). The HTTP protocol will act as the mule
The ESP will act as:
1- the medium between the User (sender) and the application
2- the application of the end-device itself. The respective relationship will be served from the async web server.
ESP32 MODULE
B-Mule Sending Sheet
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
SEND
Sender's Side
App.js
transmitting
Server.js
replying and eventually closing the channel
Application/Receiver Side
The Application will act as:
1- a dashboard as a control panel and
2- the receiver that will check the message and eventually sign it back in order to simulate a full cycle,
3- the closing intent part where the channel will get closed

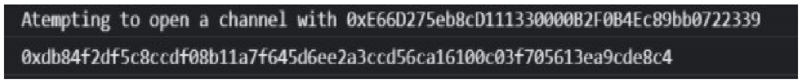
Figure 4.1: Architecture of the Demo.

With the ESP, an async server was implemented in order to be able to access it through its static IP from a mobile phone. This server will act as the application of the sender of the information. The HTTP protocol which will transmit the data can be taken as the mule (only conceptually as the mule has a real wallet address). This was done in order to visualize the whole logic better. From the module, we simulate a message delivery intent that will get captured and sent to the application through the scripts, thus simulating an interaction with a mule and a client.

The App.js is the control panel from which we can 'send' a message after an account has been connected to the test-net chain. As far as it concerns the Server.js, it can be considered the other member of the whole cycle. As mentioned in the previous chapter, the Server.js will try to decrypt the message if the hash is valid and eventually sign the latter and finalize the transaction.

It is worth mentioning that the application is implemented in a way that permits the testing of all phases from the data acquisition, to the closure of the channel and as a result the mule is represented as well in the initial interface.

More on this in the following paragraphs. As soon as the application loads, it checks whether an account has been connected to MetaMask. In case there is an active connection it will open a channel and the returned message is depicted below: After opening the channel successfully, in the upper right corner of Figure

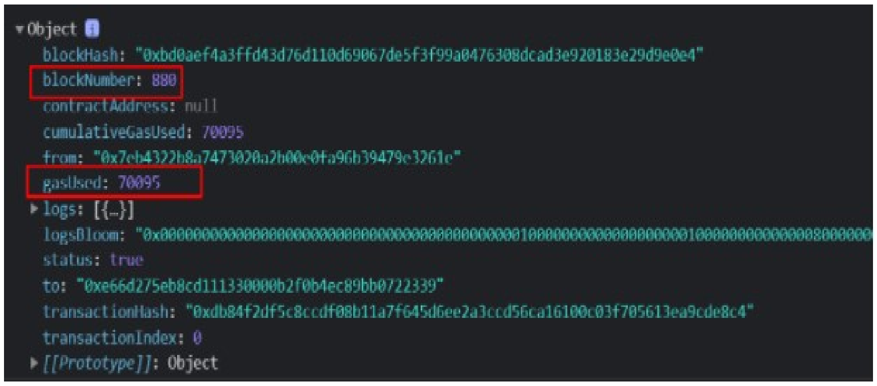


```
Atempting to open a channel with 0xE66D275eb8cD111330000B2F0B4Ec89bb0722339
0xdb84f2df5c8ccdf08b11a7f645d6ee2a3ccd56ca16100c03f705613ea9cde8c4
```

Figure 4.2: Response from Metamask.

4.4 we can see some useful data regarding the connected wallet address, the state of our balance, our approved allowance and the balance of our Mule. The data displayed in the DEVICE section can be thought of as the content of the mule. For testing purposes, there is a submit button that initiates the sending process meaning the delivery intent.

In Figure 4.3, we can see the summary of the opening process. More specifically, we can see the block number that was assigned to the channel which means that from now on until the latter closes, the channel will be identified by checking the block number and the gas used to do such an operation on the Ethereum test-net.

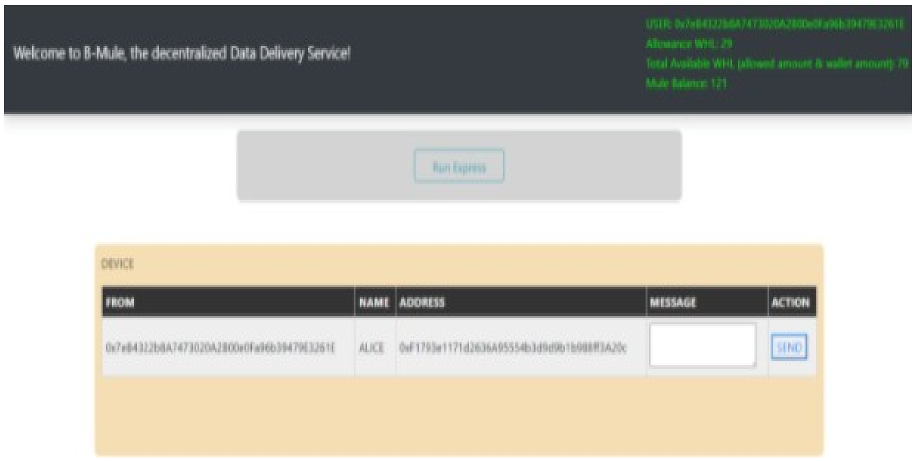


```
▾Object i
   blockHash: "0xbd0aef4a3ffd43d76d110d69067de5f3f99a0476308dcad3e920183e29d9e0e4"
   blockNumber: 880
   contractAddress: null
   cumulativeGasUsed: 70095
   from: "0x7eb4322b8a7473020a2b00e0fa96b39479e3261e"
   gasUsed: 70095
 ▸ logs: [{…}]
   logsBloom: "0x00000000000000000000000000000000000000000000001000000000000000000001000000000000000008000000
   status: true
   to: "0xe66d275eb8cd111330000b2f0b4ec89bb0722339"
   transactionHash: "0xdb84f2df5c8ccdf08b11a7f645d6ee2a3ccd56ca16100c03f705613ea9cde8c4"
   transactionIndex: 0
 ▸ [[Prototype]]: Object
```

Figure 4.3: Response from Metamask.

By clicking the SEND button, the first thing that happens is that MetaMask will ask for the signature of the data as can be seen in Figure 4.5. When the data reaches this point, it is already encrypted with the public key of the receiver. The latter is then technically sent to the mule (in this case it's the connection between the App.js and the Server.js). In order to observe what is happening through the process, we logged the results of every step. As depicted also in Figure 4.6 as soon as the sender signs the message, a complex object is created. From it, it is worth mentioning the following details:

- The block number,


Welcome to B-Mule, the decentralized Data Delivery Service!

USER: 0x7e84322b8A7473020A2800e0Fa96b39479E3261E
Allowance WHL: 29
Total Available WHL (allowed amount & wallet amount): 79
Mule Balance: 121

DEVICE

FROM	NAME	ADDRESS	MESSAGE	ACTION
0x7e84322b8A7473020A2800e0Fa96b39479E3261E	ALICE	0xF1793e1171d2636A95554b3d9d9b1b988ff3A20c		SEND

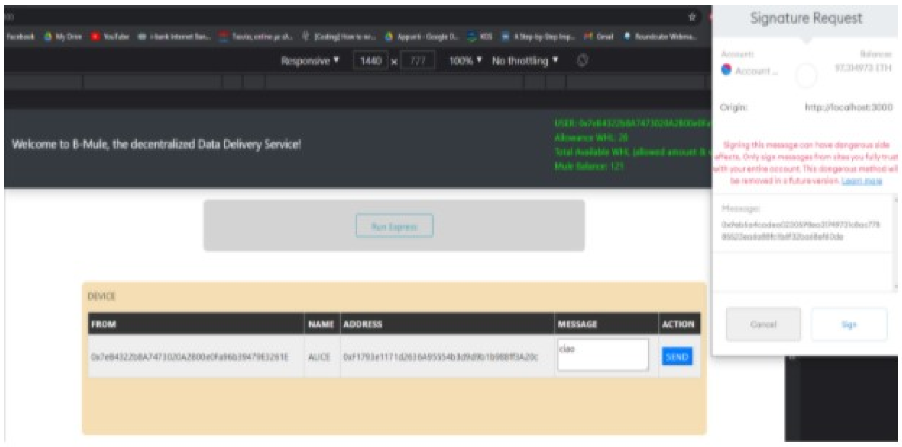
Figure 4.4: Response from Metamask.

- The cost,
- The boolean that checks whether this particular cycle has been completed,
- The balance of the sender (it's the latest after the operation)

Moreover, by taking a look inside the object we can see that there are the following structures:

- A whistle and,
- A paymentInfo

In order to keep the integrity of the application, the data inside the two objects must coincide and from Figure 4.6, we can see that this is our case meaning that the transmission of all the important data has been done correctly. As far as it concerns the receiver's signature, as mentioned earlier, it is an operation that is being carried away inside the Server.js logic. The result of the operations can be seen in Figure 4.7 where the complete log is printed in the console. The first thing to notice is the boolean attribute 'delivered' that was set to true. That means the delivery was successful and the receiver signed the message implying his agreement on the state of the transaction. The attribute *timeStamp* is critical because it serves as a filter when rewarding the mule. As already mentioned the latter will get the reward based on the latest settled transaction. If for some odd reason, one of the members does not agree upon a given state, then, if the channel closes, the mule will get paid based on the last agreed state.



Welcome to B-Mule, the decentralized Data Delivery Service!

Mule Balance: 121

Signing this message can have dangerous side effects. Only sign messages from sites you fully trust with your entire account. This dangerous method will be removed in a future version. Learn more

FROM	NAME	ADDRESS	MESSAGE	ACTION
0x7eB4322b8A7473020A2800e0Fa96b39479E3261E	ALICE	0xF1793e1171d2636A95354b3d9d9b1b988ff3A20c	ciao	SEND

Figure 4.5: Requesting the sender's signature.

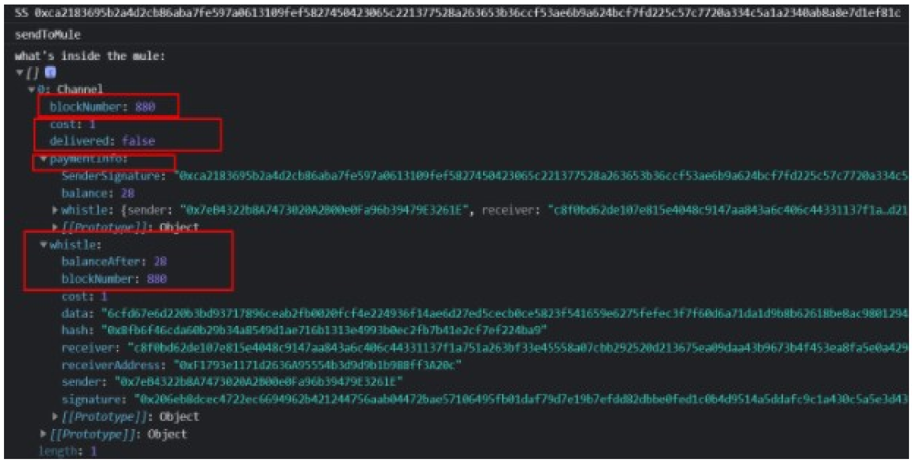
Figure 4.8 shows us the result that comes after closing a channel. The red squares indicate that the balance of the sender and the mule is now updated based on the latest transactions. The MetaMask pop up asks again if we would like to open a new channel. This was done in order to check whether the logic of the application works or not.

4.3 Results

In the following section we summarized the overall costs of an entire cycle, meaning:

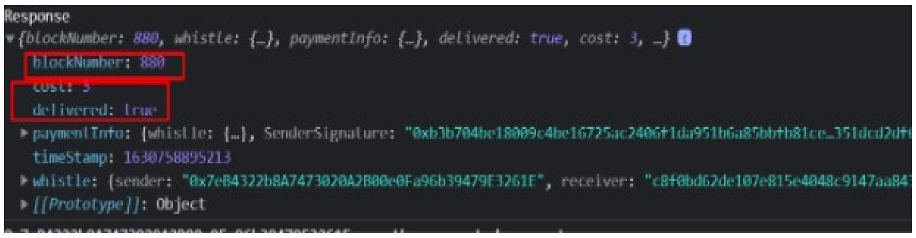
- Approve() function in order to approve the spending on behalf of the client,
- Declaring that the channel is being opened,
- TransferFrom() for the reward of the Mule,
- Uploading the settlement of the closure of the channel.

The table depicted in Table 4.1 describes the cost of opening a channel, sending a single offline message and then closing it while uploading the verdict on the blockchain.



```
SS 0xca2183695b2a4d2cb86aba7fe597a0613109fef5827450423065c221377528a263653b36ccf53ae6b9a624bcf7fd225c57c7720a334c5a1a2340ab8a8e7d1ef81c
sendToMule
what's inside the mule:
▾[]
  ▾0: Channel
     blockNumber: 880
     cost: 1
     delivered: false
   ▾paymentInfo:
       SenderSignature: "0xca2183695b2a4d2cb86aba7fe597a0613109fef5827450423065c221377528a263653b36ccf53ae6b9a624bcf7fd225c57c7720a334c5
       balance: 28
     ▸whistle: {sender: "0x7eB4322b8A7473020A2B00e0Fa96b39479E3261E", receiver: "c8f0bd62de107e815e4048c9147aa843a6c406c44331137f1a…d21
     ▸[[Prototype]]: Object
   ▾whistle:
       balanceAfter: 28
       blockNumber: 880
       cost: 1
       data: "6cfd67e6d220b3bd93717896ceab2fb0020fcf4e224936f14ae6d27ed5cecb0ce5823f541659e6275fefec3f7f60d6a71da1d9b8b62618be8ac9801294
       hash: "0x8fb6f46cda60b29b34a8549d1ae716b1313e4993b0ec2fb7b41e2cf7ef224ba9"
       receiver: "c8f0bd62de107e815e4048c9147aa843a6c406c44331137f1a751a263bf33e45558a07cbb292520d213675ea09daa43b9673b4f453ea8fa5e0a429
       receiverAddress: "0xF1793e1171d2636A95554b3d9d9b1b988ff3A20c"
       sender: "0x7eB4322b8A7473020A2B00e0Fa96b39479E3261E"
       signature: "0x206eb8dcec4722ec6694962b421244756aab04472bae57106495fb01daf79d7e19b7efdd82dbbe0fed1c0b4d9514a5ddafc9c1a430c5a5e3d43
     ▸[[Prototype]]: Object
   ▸[[Prototype]]: Object
   length: 1
```

Figure 4.6: Result of the hypothetical transmission from the sender to the Mule.

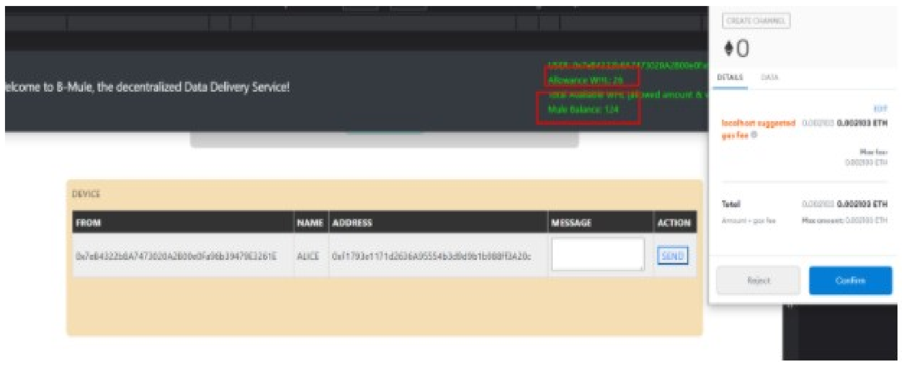


```
Response
▾{blockNumber: 880, whistle: {…}, paymentInfo: {…}, delivered: true, cost: 3, …}
   blockNumber: 880
   cost: 3
   delivered: true
 ▸paymentInfo: {whistle: {…}, SenderSignature: "0xb3b704be18009c4be16725ac2406f1da951b6a85bbfb81ce…351dcd2df
   timeStamp: 1630758895213
 ▸whistle: {sender: "0x7eB4322b8A7473020A2B00e0Fa96b39479E3261E", receiver: "c8f0bd62de107e815e4048c9147aa84
 ▸[[Prototype]]: Object
```

Figure 4.7: Data that comes from the delivery of the message.

ACTION	**GAS(Gwei)**	**ETHER**
Opening the channel + approve allowace	70095	0.002103
Closing the channel + transferFrom()	39743	0.002782
TOTAL	**109838**	**0.004885 (19.22 USD)**

Table 4.1: Actions Cost Table.


elcome to B-Mule, the decentralized Data Delivery Service!

USER: 0x7eB4322b6A7473020A2B00e0Fa

Allowance WHL: 26

Total Available WHL (allowed amount &

Mule Balance: 124

DEVICE

FROM	NAME	ADDRESS	MESSAGE	ACTION
0x7eB4322b6A7473020A2B00e0Fa9Eb39479E3261E	ALICE	0xf1793e1171d2636A95554b3d9d9b1b988ff3A20c		SEND

Figure 4.8: A cycle has closed and the Mule balance has been updated.

Chapter 5

Conclusions

In this project we tried to create a service that delivers messages securing them with the aid of the native characteristics of the blockchain. Thus, the project satisfies the initial requirements by maintaining stable off-chain transactions and guaranteeing their security and validity. With the implementation of state channels we tried to reduce the costs related to transaction fees and the waiting time since the only interaction with the chain is during the declaring intent to open a channel and the one related to the closure. We also took advantage of the native features of the BC like immutability, security and scalability in order to have a safe environment that is, in theory, tamper-proof. Another aim of the project was to render it scalable through the usage of layer 2 scaling solutions which are a good alternative when the issue is the scalability and the costs of the respective operations.

As far as it concerns our application, the implementation of the aforementioned concepts had a positive outcome since the overall costs were greatly reduced compared to what it would have been if the state channels were not implemented. In specific, let's take a look at the following 4 operations that are necessary in order to be able to interact with the application:

1. **Opening** the channel,
2. **Creating an allowance** to be spent on our behalf,
3. **Closing** the channel and,
4. **Transferring** the funds into the mule's wallet

The overall operational cost of the interaction was **19.22 USD**. This amount is independent of how many messages were exchanged between the parties. One thing to note is that the way these fees are formulated is based on the market trend as well and given the fact that our tests were made during a frenzy period with the NFTs, the fees can be considered expensive. Nonetheless, the amount of money that we would have paid if no state channels were implemented, it would have been exponentially greater. For example, a single transaction in

Ethereum (during the frenzy) could go as high as 23 USD and that is just for one transaction. Based on our application that would mean [4*23=92] USD just for the operational costs. As a result, we can see how the implementation of layer 2 scaling solutions can help overcome the cost issue when talking about the Ethereum network.

It's worth mentioning that although it seems like layer 2 scaling solutions are the only way to overcome these problems, recently Ethereum had a fork named 'London Hard Fork' which changed the core algorithm. This particular upgrade was proposed in the EIP-1559 and it introduces a new mechanism called fee burning. Basically, part of the fee is burnt and lost (making it a deflationary asset in the long run) and the remaining part is given to the miners-validators as a reward for validating the transactions block.

These features are the ones that attract more and more people thus leading to a wider adoption of such a technology. In fact, during the last year, institutions like MicroStrategy and others had adopted Ethereum and as a result invested in it. But there is also a downside. With more people, more resources are needed meaning higher fees in general and this is where layer 2 scaling solutions come into play for the time being. Up until Ethereum transitions from a PoW protocol to a PoS, layer 2 solutions like Polygon, Optimism etc, will play a major role in the institutional adoption of the blockchain technology.

Bibliography

[1] Token Economy: How Blockchains and Smart Contracts Revolutionize the Economy-Shermin Voshmgir

[2] https://ethereum.org/en/developers/docs/web2-vs-web3/

[3] V. Buterin et al., "Ethereum white paper: a next-generation smart contract decentralized application platform," First version, 2014

[4] https://ethereum.org/en/developers/docs/evm/

[5] https://finance.yahoo.com/news/ethereum-transaction-fees-hit-record-090233423.html

[6] https://eips.ethereum.org/EIPS/eip-20

[7] https://chain.link/

[8] https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44

[9] https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/state-channels/

[10] https://keccak.team/files/Keccak-reference-3.0.pdf

[11] cryptographic sponge functions, January 2011, http://sponge.noekeon.org/

[12] https://arxiv.org/ftp/arxiv/papers/1806/1806.03693.pdf

[13] https://www.techtimes.com/articles/264857/20210901/top-10-most-expensive-cryptopunks-nfts-sold