A Study of Oracle Systems for the QTUM Blockchain Eco-system.

By

Mr. George John Chavady

A Dissertation

submitted to the University of Dublin, in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Prof. Donal O' Mahony

August 2020

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university.

Signed:

_____

George John Chavady

26/08/2020

Permission to lend and/or copy

I agree that the Trinity College Library may lend or copy this dissertation upon request.

Signed:

_____

George John Chavady

26/08/2020

Acknowledgments

First and foremost, I would like to thank God Almighty for helping me complete my research. Next, I would like to express my sincere gratitude to my supervisor, Prof. Donal O'Mahony for his continuous support, guidance, encouragement and expertise during my Masters thesis.

I also thank my parents Mr. John Cherian and Mrs. Aneyamma John, for all their support and love during the period of my course. I am very grateful to Mr. Manoj George, my uncle for his support during my stay in Ireland. My brother Paul John and Elizabeth John have always been a pillar of support.

George John Chavady

University of Dublin, Trinity College

August 2020

Table of Contents

Chapter 1

Introduction

The world presents to us a number of use cases that require decentralization. For example, sending an official e-mail is a process followed by most businesses. An important email can be prone to fraud when human personnel is in-charge. This process is automated to avoid centralization by a company. Though the process has been decentralized, the data that is used by the automated process can be altered to perform fraud, if someone has access permissions to do so. The email sent can also be deleted from the system, so as to destroy evidence. The solution is to have a system that combines both decentralization (or automation) with immutability.

Blockchain is a technology that combines the qualities of decentralization and immutability. It achieves decentralization using a peer-to-peer network to connect all the users of the system. Each communication in the network is recorded as a transaction and stored permanently in the network. It uses various types of consensus protocols to reach an agreement on whether a transaction is valid. There is no single entity that fully controls the operation in the blockchain. The cryptography techniques and incentive schemes are carefully designed to help create a distributed ledger shared by all the members of the network and is immutable.

By design, the blockchain network has no central authority with the ability to unilaterally approve invalid transactions or roll back and alter the state of previous transactions. The distributed ledger stored on every users machine is an append-only log for storing data in an immutable manner. A new transaction is only added at the end of the ledger, and no previous transactions can be modified. Each user possesses a private key that is used to authorize transactions, preventing forgery. Moreover, a new transaction is not immediately added to the ledger but is bundled together with other pending transactions into a block. This is done to prevent double spending attacks which refers to the same currency used more than once in several transaction. [1]

There are several blockchains such as Bitcoin, Ethereum, Quorum, Hyperledger etc. Ethereum was the first blockchain that supported the implementation of smart contracts. A Smart Contract is a program deployed in a distributed network that can acquire outside information and update its internal state automatically. [4] The ultimate goal of the smart contract is to use computer intelligence instead of human intelligence to make the system more efficient and credible.

1.1 Motivation

The true potential of a smart contract can be exploited when data can be made available to the decentralised system in which the smart contract is deployed from the outside world. An oracle is an entity which provides trustworthy information from a service that is external to a blockchain. E.g., Suppose that George and Paul place a bet on who the winner of the US presidential election will be. George believes that the Republican candidate will win, while Paul believes that the Democrat will be the winner. They agree on the terms of the bet and lock their funds in a smart contract, which will release all the funds to the winner based on the results of the election.

Since the smart contract is passive and cannot interact with external data, it has to be supplied with the necessary information by an entity that has access to the outside world. This entity is called an oracle. After the election is over, the oracle queries a trusted API on behalf of the parties involved in the contract to find out which candidate has won and relays this information to the smart contract. The contract then sends the funds to Alice or Bob, depending on the outcome, more importantly depending on the data sent by the oracle service. Without the oracle relaying the data, there would not have been a way to settle this bet by transfer of funds. It has to be noted that the efficiency of the oracle service is quintessential to the proper working of the smart contract. If the service opts to send data that is false, this could result in funds being transferred to the wrong party. [9]

Several oracle solutions are currently available in the market. Oraclize.it [6] fetches data from a specified web source and publishes it to a blockchain application. They also maintain cryptographic proofs which show that the information originated from the correct source. Town Crier [7] is another oracle, which works in a similar fashion. It makes use of Intel Software Guard Extensions [8] to protect against tampering.

Oracles are essential implementations in numerous applications where there is a need for data that is external to be brought into the blockchain. The current oracles provide a solution without robust security guarantees that the blockchain provides. These therefore are impediments to security and could possibly become centralized points-of-failure or attack. Thus, the oracles in the Blockchain eco-system remain a subject of research and innovation. In this work we explore the existing literature and state of the art in blockchain, decentralized applications and oracles in block chain. We also aim to work on designing an Oracle system that improves upon existing architectures.

1.2 Research Question

Can we find an improvement upon existing oracle implementations in the blockchain eco-system?

1.3 Objectives

1. A review of the Bitcoin core and its various features.
2. To gain an understanding of the Ethereum blockchain and decentralized applications.
3. Understanding and implementing smart contracts using Solidity, which is an Object-Oriented Programming Language that targets the Ethereum Virtual Machine.
4. Examine work on Oracles – principally on the Ethereum blockchain.
5. Research on the QTUM blockchain eco-system.
6. Innovate and design an Oracle system, which is an improvement from the existing implementations.

1.4 Challenges

1. Lack of substantial literature on Oracles and their implementations.
2. Bringing innovation to the existing work on Oracles.
3. Insufficient documentation for oraclize.it service.
4. Fewer implementations of smart contracts and oracles available on the web or other resources.
5. Compatibility of solidity compilers was often an issue. Hence had to avoid using compilers having versions greater than 4.25.

1.5 Overview

The use of oracles to send data that is external, into the blockchain is critical to the proper functioning of smart contracts deployed within a blockchain. The concept of oracles is an interesting development in the blockchain industry. In order to gain a proper understanding of the working of oracles, it is important to understand the state of the art in blockchain. This work consists of review of Bitcoin and Ethereum blockchain concepts. Bitcoin blockchain uses a design that implements decentralization and immutability to manage the bitcoin currency. Unlike Bitcoin that allows only simple operations to be performed to ensure security, Ethereum also allows code of complex nature to be executed on the blockchain. Further, we understand the QTUM blockchain eco-system that uses the best of both Bitcoin and Ethereum blockchains. QTUM provides a Turing-complete blockchain stack that can execute smart contracts and decentralized applications and, uses the Ethereum Virtual Machine (EVM). However, in contrast to Ethereum, QTUM is built upon Bitcoin's Unspent Transaction Output (UTXO) model and employs a Proof-of-stake consensus mechanism that is more practical for business adoption. A transaction in the blockchain consists of inputs and outputs. The outputs that have not been spent yet are referred to as UTXOs. Proof-of-stake means that the creator of the next block is chosen based on the held wealth in crypto-currency. We examine the various oracles currently available in the market and categorises them as centralized and decentralized. The work also explains implementations of smart contracts and oracles on the Ethereum blockchain. An innovation from the existing implementations of oracles has also been discussed and designed which is the contribution to the literature on Oracles for blockchain.

1.6 Thesis Structure

The thesis is organized as follows. Chapter 2 presents the literature review on Blockchain, Bitcoin Core, Ethereum and Smart Contracts. Chapter 3 examines the work on Oracles in blockchain. Chapter 4 focuses on the design aspects of various Oracles that are currently available in the market. Chapter 5 discusses the implementation of smart contracts and oracles that interact with the contracts deployed in the Ethereum blockchain. It also discusses a few innovations with respect to oracles and their design. Chapter 6 evaluates the newly proposed design of oracles

and checks for feasibility. The last section consists of conclusion and future works or research.

Chapter 2

Literature Review

2.1 Blockchain

Todays' payment systems facilitate an exchange of currency between two entities namely a payer and a payee. Apart from the payer and payee, a payment system typically involves two more entities that act on behalf of the parties involved in the transaction. One entity manages funds on behalf of the payer, known as the issuing bank (or issuer), and another entity that maintains an account for the payee, known as the acquiring bank or acquirer.

The operations of a typical cash-like system (traditional banking system) are depicted in Figure 2.1. In a cash-like system, the payer's account is charged a fee before the actual payment takes place. Transactions require that a payment be made to the intermediary bank which usually is a percentage of the amount that the payer wishes to transfer. Businesses need to shell out from their profits towards high banking fees when the total amount transferred and the number of transactions increase. [7] Further, the amount is transferred through intermediaries and organisations that maintain their own logs with unrestricted access to alter them. Blockchain improves the payment system by ensuring and assuring parties of security through immutability, higher transfer speeds, lower conversion fees and a trustless service. This is achieved first and foremost by eliminating the need for centralized control (e.g., by banks) to transfer funds and perform third-party authorizations through the implementation of a shared distributed ledger. The distributed ledger is an append only log that stores all the transactions that occur, with a guarantee that they cannot be altered. Security is enforced by maintaining a hash of the previous block within every block such that the genesis (or the first) block can be verified. These transactions are made public so that

all the stake holders could access them and check for integrity if needed. It requires only the parties involved to authorize using a private key.

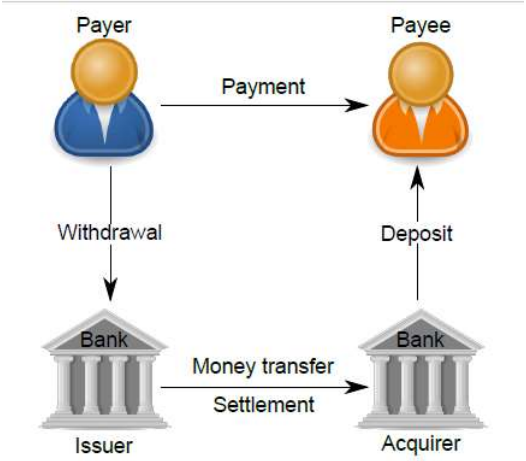

Figure 1: Cash-like Payment System Architecture

More formally, a blockchain can be defined as a system that decentralizes the working of an application where there is no single entity that controls the operation of the blockchain. The network has no central authority that is able to unilaterally approve invalid transactions or manipulate the state of the system through any means aside from normal submission of transactions for processing. The system uses a distributed ledger for storing the transactions. A new transaction can be added only at the end of the ledger, and no previous transactions can be modified. Moreover, a new transaction is not immediately added to the ledger but is bundled together with other pending transactions into a block. [1]

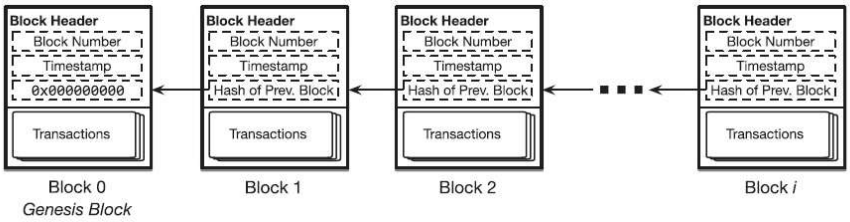

Figure 2: A Simple Blockchain

Blockchain as a Ledger

Blockchain is a distributed ledger technology which consists of three elements. First, the ledger is continuously amended and persistent, where all transactions effected on the blockchain are perpetually stored. Secondly, a blockchain is a distributed peer to peer ledger, where an entire copy of the blockchain is stored on every node in the network. If new transactions are affected, majority of nodes in the network must verify the legitimacy of the effected transaction and, if confirmed, every node is updated with the transaction. With this distributed authentication process, the core feature of blockchain which is the lack of a central entity or intermediary is facilitated. And thirdly, blockchain is asymmetrically encrypted and requires private and public keys to make a transaction. [9]

For example, the Bitcoin technology uses a ledger to record the transactions and also track the ownership of tokens called Bitcoins. The tokens are distributed among nodes that represent accounts that are each uniquely identified by a public key. Bitcoin associates a public key with a balance in Bitcoins. Each item added to Bitcoin's blockchain-backed ledger serves as a record of a transaction which denotes a transfer of Bitcoins from one public key to another.

2.2 Bitcoin

Bitcoin is a blockchain that supports the working of the bitcoin cryptocurrency. In Bitcoin, payments are performed by issuing transactions that transfer Bitcoin coins, referred to as BTCs, from the payer to the payee. Both payers and payees are peers in the network and are referenced in transactions by Bitcoin addresses. Each address of a peer in the network is mapped to a unique public/private key pair, which is used to transfer ownership of BTCs among addresses. A private key in bitcoin is a large random number between 1 and 2 ^ 256, which could take an attacker billions of years to try all possible combinations. A hash function on the private key is used to generate a public key. Users are able to generate public/private key pairs and the bitcoin address is a fixed length binary quantity of 26 to 35 alphanumeric characters which is derived using the public key.

The bitcoin blockchain operates on top of a loosely connected peer-to-peer network, where users can join and leave the network as they wish. The peers or users connect to the blockchain network by requesting a list of current bitcoin peer addresses from

the Domain Name System seeds. DNS is a protocol that maps domain names to IP addresses of users connected to the network. For example, the domain name www.oracle.com translates to the addresses 91.182.212.36 (IPv4). [15] Each Bitcoin address is computed from an Elliptic Curve Digital Signature algorithm (ECDSA) public key, for which the address owner knows the corresponding private key using a transformation based on hash functions. Hashes are one-way functions which allow the computation of the address using the public key but makes it infeasible to retrieve the public key from the address alone.

A Bitcoin transaction is created by digitally signing a hash of the transaction where the coins were last spent, together with the public key of the recipient. Transactions take as input the output of the previous transaction that spent the same coins and the output is a list of addresses that can collect the coins transferred by the transaction. A transaction output can only be redeemed once, after which the output is no longer available to other transactions. This process is facilitated in bitcoin by the implementation of Unspent Transaction Output model (UTXO, see section 2.2.1). The recipient in the transactions is able to redeem the coins using his private key that matches the public key used in the transaction creation process. Once ready, the transaction is signed by the user and broadcast in the P2P network. Any peer can verify the authenticity of a BTCs by checking the chain of signatures using public keys.

The difference between the input and output amounts of a transaction is collected in the form of fees by Bitcoin miners. Miners are peers that participate in the generation of Bitcoin blocks. These blocks are generated by solving a hash based proof-of-work (PoW) algorithm (see section 2.2.2). More specifically, miners must find a nonce value that, when hashed with additional fields (the Merkle hash (see section 2.2.3) of all valid transactions, the hash of the previous block), the result is below a given target value. If such a nonce is found, miners then include it in a new block, thus allowing any entity to verify the PoW. The underlying proof-of-work allows different miners to find the nonce value and create different blocks nearly at the same time which is when a fork in the blockchain occurs. Forks are inherently resolved by the Bitcoin system through a mechanism where the longest blockchain that is backed by the majority of the computing power in the network will eventually prevail. [7]

2.2.1 UTXO Model

Bitcoin transactions use outputs from previous transactions as inputs in the construction and execution of a new transaction. For example, consider that Alice wants to send Bob 1 bitcoin and the transaction fee required is 0.25 bitcoins. Such a transaction could have the following inputs:

Input 1 – 0.5 BTC
Input 2 – 0.25 BTC
Input 3 – 0.5 BTC

The inputs considered above were outputs from the previous transaction. Considering the transaction fee of 0.25 BTC, the output of the transaction, i.e. the number of bitcoins Bob would actually receive, would be:

Output 1 – 0.5 BTC
Output 2 – 0.5 BTC

Bob would thus receive 1 bitcoin at the end of the transaction. The output of a transaction can either be classified as an unspent transaction output (UTXO) or be classified as spent transaction output. The unspent transaction output later becomes an input for transactions performed by Bob. For transactions such as that of Alice's to be valid, they must only use unspent transaction outputs as inputs. This validity is checked for by the implementation of a UTXO set.

More formally, an unspent transaction output (UTXO) is an abstraction of electronic money that can be used for future transactions. Each UTXO represents a chain of ownership implemented as a chain of Digital Signatures where the owner signs a message (transaction) transferring ownership of their UTXO to the receiver's Public Key. The receiver node is able to unlock the value sent by the payer using its private key that matches the public key specified within the transaction.

UTXO Set

The function of the UTXO set is that of a global database that shows all the spendable outputs that are available to be used in the construction of a bitcoin transaction. When a new transaction is initiated by a user, it uses an unspent output from the UTXO set of the user, resulting in the set shrinking. On the contrary, when a new unspent output is created (for example, through mining), the UTXO set will grow.

Bitcoin full nodes download every block and transaction to check them against Bitcoin's consensus rules. [13] They are required to track all the unspent outputs in existence on the Bitcoin network in order to ensure that a user is not attempting to spend bitcoins that have already been spent, i.e. a double spending does not take place. To prevent double spending and fraud, inputs on a blockchain are marked as spent when a transaction occurs, while at the same time, outputs are created in the form of UTXOs. These unspent transaction outputs may be used their owners (the holders of private keys) for their future transactions. [14]

A user’s bitcoin balance is the sum of all the individual outputs that can be spent by their private key. Therefore, when a user initiates a transaction, the outputs from the user’s UTXO set is used. All the unspent outputs must entirely be consumed when a transaction is being conducted, with change being sent back if the total value of the outputs is larger than the value of the transaction.

For example, if a user has a UTXO worth 10 bitcoins, but only requires 2 bitcoins for their transaction, then the entire 10 bitcoins is sent with two outputs being produced:

Output 1 – 2 BTC payment to the recipient
Output 2 – 8 BTC payment back to the user’s wallet as change

A transaction consumes previously recorded unspent transaction outputs and creates new transaction outputs that can be used for a future transaction. This allows bitcoins to move from one owner to another, with each transfer consuming and creating UTXOs in a series of transactions. [12]

Components in a Transaction Output

scriptPubKey is a locking script placed on the output of a Bitcoin transaction that requires certain constraints to be satisfied so that a recipient could spend the bitcoins

that have been transferred to them. Conversely, scriptSig is the unlocking script that satisfies the conditions placed on the output by the scriptPubKey, and this is what allows the bitcoins to be spent.

Using the previous example, in order for Bob to spend the bitcoins received from Alice, each output will contain a locking script, scriptPubKey, which must first be satisfied by the unlocking script, scriptSig which uses Bob's private key.

To illustrate, when Alice decides to initiate her transaction with Bob, the outputs that Bob receives contains bitcoins that can be spent only when the conditions laid out by the attached scriptPubKey are satisfied. When Bob decides to spend these outputs, he creates an input that includes an unlocking script, scriptSig, that must satisfy the conditions that Alice placed on the previous outputs using scriptPubKey before he can actually spend them. [12]

2.2.2 Proof of Work

As discussed already, the bitcoin blockchain is maintained by a peer-to-peer network. The transaction required by a user is executed by the peers in the network. When users add a new entry to a blockchain's ledger, they submit a transaction to a node in the network using a Remote Procedure Call (RPC) protocol. The member broadcasts the transaction to the rest of the network for inclusion in a future block. Similarly, a user may submit a query to a network member about the contents of the blockchain's ledger. The parties involved in a transaction, such as the sender and receiver of Bitcoins on the Bitcoin blockchain do not have complete control of the execution of that transaction. Instead, the task falls to the members of the network who validate the transactions and the miners include the transaction within a block during the block creation process.

The members of the network or the peers can be classified into full nodes and miners. Full nodes are network members that own a full copy of the blockchain, containing every block and thus every ledger item, and keeps this copy synchronized with the latest updates to the blockchain by continuously monitoring the network for notification of new blocks. They help broadcast the transactions of users to the rest of the network. Full nodes commit computation and storage resources to this purpose. Also, by retaining a copy of the blockchain users do not have to trust an intermediary

service to query the blockchain's state or submit transactions on their behalf. A subset of the members within a blockchain's peer-to-peer network not only maintain copies of the blockchain, but also actively construct and propose new blocks to be added to the chain. This process of constructing and adding new blocks to the network is known as mining, and these members are therefore referred to as miners. Miners must follow a certain protocol to ensure, the property of consensus where all members of the blockchain network together decide on the new block that is to be added to the chain and have an identical view of all previous blocks. This means that all blockchain copies are identical across the network. While there is an additional computational cost to assembling blocks and participating in a consensus protocol, full nodes may choose to run miners, because they have a vested interest in the successful operation of the blockchain or because of more explicit incentives.

In Bitcoin and several other blockchains, miners follow a proof-of-work algorithm (see Figure 3) to determine which miner appends the next block in the chain. The main rationale for using this algorithm is to prevent miners from immediately appending a newly prepared batch of transactions as a new block on the chain. If this were permitted, then many miners could continuously and simultaneously grow the chain, making it difficult to determine a globally recognized ordering of blocks, which is required to form a unified view of the blockchain's state. The system is designed such that each newly appended block must also contain a random value called a nonce, such that a cryptographic hash of the block's contents, including the numerical value of the nonce, falls below an upper threshold 't'.

Because a sound cryptographic hash function cannot be inverted, the only means of discovering a nonce satisfying this constraint is through brute-force search. This search process is the work and the satisfying nonce is the proof of this work. The first miner to find a proof appends the next block to the chain.

The following are the steps involved in mining a block using the proof-of-work consensus algorithm –

(1) Choose a set of pending transactions that have been received from the peers on the network but have not yet been included in any of the previous blocks and bundle these transactions as a payload p.

(2) Search for a nonce n that, when concatenated with p, produces a cryptographic hash that does not exceed a specific threshold. So we find, H(p. n) <= t for some bit string t.

(3) If some other valid block is received before n is found, append that block to the chain and return to Step 1.

(4) When the proof of work n is found, broadcast the new block which includes n, to the network. Return to Step 1.

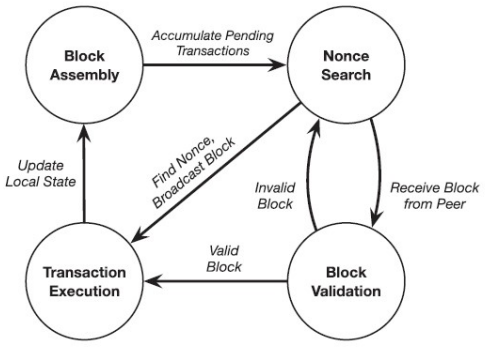

Figure 3: Proof-of-work consensus

Proof-of-work can be considered a repeated lottery that determines which miner is allowed to append the next block to the chain. All other miners validate and accept the new block before moving on to the next round of the consensus i.e., a new lottery. A miner's odds of winning a lottery round (its degree of influence over the operation of the blockchain) are proportional to the rate at which it can test nonce values in search of a valid proof of work. Therefore, a miner's influence is tied to its computing power. The tying influence of the result to the miner's computing power also gives proof-of-work consensus resilience to Sybil attacks. A Sybil attack is a technique in which an adversary disguises as many users of a system to gain control over the system.

The quantity t is a bit string that represents an upper bound on the output of the hash function on the cryptographic hash of the block's contents, including the nonce used to produce the proof of work. This threshold is controlled by an adaptive and time-varying parameter known as the difficulty of the mining process. A smaller t value reduces the number of values that can serve as a valid proof of work, while a larger t

value increases the number of such values, thus increasing the probability of finding a nonce. Therefore, mining difficulty can be defined as the expected number of values that must be tested before any of the network's miners succeeds in finding a proof of work. It is adjusted to keep the expected time delay between two successfully mined blocks constant even as the collective computing power of a blockchain's peer-to-peer network fluctuates over time as nodes join and leave the network. [1]

Forks

Proof-of-work consensus is analogous to a lottery system that is used to select the next block in the chain. Each new block is chosen non-deterministically and there can be situations where different miners find a nonce and append a block of transactions to the chain nearly the same time which leads to a split view of the blockchain's state among the members of the network, known as a fork in the chain (see Figure 4).

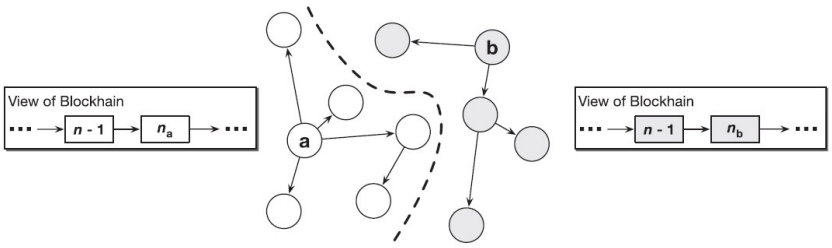

Figure 4: Fork in a blockchain

Consider for example two miners a and b in the blockchain network, and each is able to find a valid nonce that satisfies the proof-of-work consensus protocol nearly the same time. Both a and b broadcast the newly created block together with the nonce to the network. As the new blocks propagate through the blockchain's peer-to-peer network, one group of nodes accept and append a's block to their local copy while another group of nodes append b's block to their local copy. The same is the case with miners as well, where there is one group of miners that accept the block created by a and a second group of miners that accept b's block. The hashing power of the network is now split into two competing groups, thus making the network vulnerable to attacks.

This problem is resolved by adding a simple rule to the proof-of-work consensus protocol i.e., when a node observes a fork in the blockchain, it must always follow the

longer of the two chains, i.e., the chain that contains a greater number of blocks. Every node should treat the longer fork as the canonical state of the chain, and every miner should append new blocks onto the head of this chain. One of the two forks will have its first block adopted by the group of nodes in the network with more hashing power. This subset of the network will then be able to mine and append new blocks at a faster rate than the nodes backing the fork with lesser hashing power. By the very fact, it will become longer over a period of time, and this disparity will only keep growing as more nodes identify the longer fork and abandon the shorter fork. Finally, it would be the longer fork that prevails.

The blockchain's structure where each block contains a hash of its previous block, and the possibility of forks to occur has led to the concept of confirmations. When a transaction is included in what appears to be the newest block on the chain, it is not yet certain that this block will become part of the canonical chain, and therefore that the transaction will finally become a part of the network. There is a probability that a block turns out to be part of an eventually abandoned fork. Moreover, the more successors a block has in the chain, the more resistant those block's transactions are to attack by adversaries. This is because an attacker must have the hashing power necessary to force the network to roll back all of a block's successors before it can alter the contents of the block itself. Therefore, many blockchain applications will not consider a transaction immutable until a sufficient number of blocks have been appended as successors to the transaction's block. Each successor reduces the probability that the transaction is manipulated by an attacker or discarded as part of an abandoned fork. [1]

2.2.3 Merkel Trees

A Merkle tree is a data structure which constitutes a hash tree where the root hash of a particular node (leaf nodes) is converged at by a sequence of hash operations along the path that converges that converges at the root node. This allows us to verify the authenticity of a particular transaction and that it has not been altered (see Figure 5). They help in building cryptographic accumulators which helps us verify whether given transaction belongs to a block in a blockchain.

Therefore, a Merkle tree can be defined as a binary tree in which the data is stored in the leaf nodes. More specifically, given a set X which contains n elements. Each of the elements 1, 2, … , n are assigned to the leaf nodes of the binary tree. Suppose a[i, j] denotes a node in the tree located at the ith level and jth position. Here, the level refers to the distance to the leaf nodes and it is evident that the leaf nodes are located at distance 0. The positions numbered starting from 0 within the level and are considered from the left-most position. For example, the leftmost node of level 1 in a tree a, is denoted by a[1, 0]. The nodes at a distance one or more from the leaf nodes are considered to be intermediate nodes and they are computed as the hash of their respective child nodes i.e., a[i+1, j] = H(a[i, 2j], a[i, 2j+1]), where H(X) refers to the cryptographic hash of X, where X represents a set of two nodes. Given below is an example of a Merkle tree (see Figure 5) accumulating eight elements, where a30 is referred to as the Merkle root and serves as the proof for all the content in the leaf elements U0, …, U7. To prove that an element U3 belongs to the set X containing elements U0, …, U7 we need to prove that the hash of the elements in the path from U3 (in our case a[0,3]) to the root a[3, 0] converges at the root. To verify this, intermediate nodes a[0,2], a[1, 0] and a[2, 1] are needed and these nodes form the sibling path of U3. Given n leaves, time complexity for constructing the tree is O(n) and the time required for proving the membership of an element is O(log(n)).

The functions used in the implementation of a Merkle tree are:

D ← Acc(X) – This function accumulates the elements of a set X into a digest whose value is stored in the root node. This can be used to prove that the exact set X is correctly accumulated in the root which stores the digested hash value.

Pm ← ProveM(X, x) – Given a set X and element x belongs to X, this algorithm outputs a proof of membership Pm asserting that x belongs to X. Pm consists of the sibling path of x in the modified Merkle tree and the root which stores the digest.

Verify m(D, x, Pm) – Given Pm, an element x, its sibling path and the root, this algorithm outputs true if and only if the hash on the sibling path of X converges at the value in the root node.

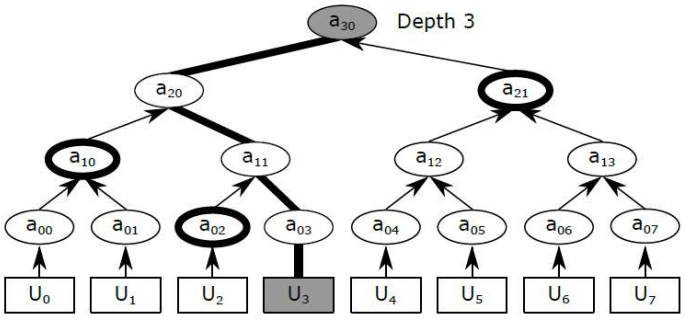

Figure 5: A Merkle tree of depth 3, accumulating eight elements U0, ..., U7

The "hash" of a block is a hash function applied on the block header which contains roughly 200-byte piece of data which includes a timestamp, nonce, previous block hash and the root hash of a data structure called the Merkle tree storing all transactions in the block. In a Merkle tree, the intermediate nodes is the hash of its two children and a single root node which is also formed from the hash of its two children, representing the "top" of the tree. The purpose of the Merkle tree is to facilitate the verification of data in the leaf nodes by the using only the header of a block. Here, the hashes propagate upward, where if an adversary attempts to swap in a fake transaction into the bottom of a Merkle tree or alter its contents in any manner, this change will cause a change in the node above, all the way until the root of the tree and therefore the hash of the block, causing the protocol to register it as a completely different block. The altered block becomes invalid since the nonce value does not satisfy the hash function applied on the contents of the newly altered block. [18]

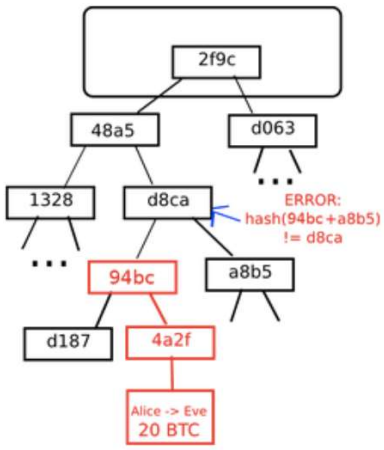

Figure 6: An attempt to change the Merkel Tree

2.3 Ethereum

Ethereum is a blockchain like bitcoin, but also has certain additional features that helps it extend its scope beyond just the management of cryptocurrencies. It shares many of the common elements such as a peer-to-peer network connecting participants, a Byzantine fault–tolerant consensus algorithm for synchronization, the use of cryptographic primitives such as digital signatures and hashes, and a digital currency (ether). Yet, the purpose of the design and construction of Ethereum is different from those of the open blockchains that preceded it, including the Bitcoin. Ethereum's purpose is not primarily to be a digital currency payment network. While the digital currency ether is necessary for the successful operation of Ethereum as a blockchain, ether is only intended as a utility currency to pay for use of the resources on the Ethereum platform such as memory and computation.

Unlike Bitcoin, which has a very limited scripting language (a set of opcodes), Ethereum is designed to be a general-purpose programmable blockchain that runs a virtual machine capable of executing code of arbitrary and unbounded complexity. Bitcoin's Script language is, intentionally, constrained to simple true/false evaluation of spending conditions. Whereas, Ethereum's language is Turing complete (see section 2.3.3), meaning that Ethereum can function as a general-purpose computer that is able to execute complex code. The idea was that by using a general-purpose blockchain like Ethereum, one could develop applications on a blockchain without having to implement the underlying mechanisms of peer-to-peer networks, blockchains, consensus algorithms, etc. The Ethereum platform is designed to abstract the complex details of the blockchain and provide a secure programming platform for developing decentralized blockchain applications. Unlike the bitcoin that tracks only the state of currency ownership Ethereum tracks the state transitions of a general-purpose data store. This store that can hold any data in the form of key–value pair. A key–value data store holds values required by a program, each referenced to by its corresponding key. For example, the value "04-09-1992" is referenced by the key "DoB". It is similar to databases used by today's modern applications which contain a set of records that are uniquely identified by a key value. Ethereum provides memory that stores both data and code (usually used to manipulate the state of the data), and it uses the Ethereum blockchain to track how the state of the memory changes over

time. Like with a general-purpose computer, Ethereum can load code into its state machine and execute that code, storing the resulting state changes in its blockchain. This helps us code logic and constraints and store them on the block chain, which can be executed by members of the network. This set of data and executable code on the blockchain is referred to as a smart contract which is discussed in detail in the coming sections. [19]

2.3.1 Accounts in Ethereum

Similar to Unspent Transaction Output Model (UTXO) in Bitcoin, in Ethereum, the state is made up of objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts. An Ethereum account contains four fields:

- The nonce, a counter used to make sure each transaction can only be processed once
- The account's current ether balance
- The account's contract code (optional)
- The account's storage (empty by default)

"Ether" is the crypto-fuel or cryptocurrency of Ethereum and is used to pay transaction fees. In general, there are two types of accounts in the Ethereum blockchain: externally owned accounts, controlled by private keys, and contract accounts, controlled by their contract code. An externally owned account has no code, and one can send messages from an externally owned account by creating and signing a transaction using their private key, where as in a contract account, every time the contract account receives a message its code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn.

Contracts in Ethereum should not considered as something that is compiled and executed rather, they are more like "autonomous agents" that live inside of the Ethereum blockchain and run on the Ethereum virtual machine. A specific piece of code is executed when called by a message or transaction from either an externally owned account or a contract within the blockchain. The contracts have direct and full control over their own ether balance and their key-value store used to store persistent

data usually required by the functionality implemented within their code. Thus, a contract is a decentralized and immutable piece of code together with data that can be used to implement a specific requirement. [18]

2.3.2 Messages and Transactions

The term "transaction" is used in Ethereum to refer to the signed data package that stores a message, to be sent from an externally owned account. It represents a message that ideally includes data and currency. The currency either constitutes of the amount transferred to the recipient or transaction costs. A transaction contains the following parts:

- A recipient address
- A signature used to identify the sender
- An amount in ether (transferred from the sender to the recipient)
- A data field (optional)
- A STARTGAS value, represents a limit on the number of computational steps a transaction can take
- A GASPRICE value, which is a fee the sender pays for each computational step and is usually measured in bytes.

The first three are standard fields usually present in any blockchain. The data field is null by default, but the virtual machine has an opcode which a contract can use to access the data. For example, if a contract is functioning as an on-blockchain betting service, then it may interpret the data being passed to it as containing two "fields", where the first field contains the value of an outcome and the second field is the value staked by the better. The contract reads these values from the message data and places them in the key-value data store.

The STARTGAS and GASPRICE fields are important for Ethereum's anti-denial of service model architecture. In order to prevent accidental or adversarial infinite loops and the resulting wastage of memory and computational resources, each transaction is required to set a limit on how many computational steps of code execution is allowed. The fundamental unit of computation is "gas". In Ethereum a computational step usually costs 1 gas, but some operations cost higher amounts of gas because they are

more computationally expensive. There are situations where the amount of data stored as part of the state of the transaction is large and a fee of 5 gas is charged for every byte. The purpose is to discourage an adversary who is financially rational from consuming computation, bandwidth and storage on the blockchain network. Therefore, a transaction that consumes resources in the network must pay a gas fee which is proportional to the resources consumed. [18]

Messages

Contracts can send "messages" to other contracts. Messages in Ethereum refer to objects that contain data and value (in ether) which are passed between two accounts. A message is usually created when contracts interact with each other or by a transaction. A message contains the following fields:

- The sender of the message (implicit)
- The recipient's address
- The amount of ether to be transferred
- A data field (optional)
- A STARTGAS value

We notice, a message is similar to a transaction, except that it is produced by a contract and not by an externally owned account, as is the case with transactions. A message is produced when a contract's code executes the CALL opcode. Like with a transaction, a message thus leads to the recipient account (a contract account) running its code. Thus, contracts can be made to interact with other contracts in exactly the same way that externally owned accounts can.

The gas required when making a transaction is the total gas required by the network for the consumption of its resources, which even includes all sub-executions present within that transaction. For example, if an external actor A sends a transaction to B with 100 gas, and B consumes 60 gas before sending a message to C, and the internal execution of C consumes 30 gas before returning, then B can spend another 10 gas before running out of gas. [18]

2.3.3 Turing Completeness in Ethereum

The term ‘Turing complete’ refers to an English mathematician Alan Turing, who is considered the father of computer science. In 1936 he created a mathematical model of a computer consisting of a state machine that manipulates symbols by reading and writing them on sequential memory. Turing later provided a mathematical foundation to answer questions about universal computability, meaning whether all problems are solvable. He proved that there are classes of problems that are not computable. Specifically, he proved that the halting problem is not solvable. Halting problem refers to the question whether it is not possible, given an arbitrary program and its input, to ascertain whether the program will eventually stop running. Ethereum’s ability to execute a stored program in a state machine called the Ethereum Virtual Machine (EVM), while reading and writing data to memory makes it a Turing-complete system and therefore a Universal Turing Machine. The EVM is able to compute any algorithm, given the limitations of finite memory and hence prone to the halting problem.

The innovation in Ethereum’s is that it combines the general-purpose computing architecture of a stored-program computer with a decentralized blockchain. It thus creates the so called ‘world computer’ that is distributed across all the nodes in the network. In Ethereum, programs run ‘everywhere’ i.e., in each node, yet maintain a common state or behaviour which is enforced by the consensus rules of the blockchain. However, Turing completeness is very dangerous, particularly in open access systems like public blockchains, because of the halting problem. For example, modern printers are Turing complete and can be given files to print that send them into a frozen state after which they cannot be used to perform other operations. The fact that Ethereum is Turing complete means that any program of any complexity can be computed by Ethereum. But that flexibility brings issues relating to security and resource management. An unresponsive printer can be restarted and continued to be used by the network. But the same is not possible with a public blockchain.

Implications

Turing proved that we cannot predict whether a program will terminate by simulating it on a computer. More specifically, we cannot predict the path of a program without running it. Turing-complete systems can run into ‘infinite loops’ and ultimately results in the usage of an enormous amount of resources. It is trivial to create a

program that runs a loop that never ends. But unintended never-ending loops can arise without warning, due to complex interactions and constraints specified within the code. In Ethereum, this poses a challenge: a smart contract can be created such that it runs forever when an externally owned account or another contract in the blockchain attempts to invoke it. This kind of a situation is termed, a DoS attack. In a world computer, a program that abuses resources gets to abuse the world's resources. How does Ethereum constrain the utilization of resources by a smart contract if it cannot predict the resource usage in advance?

In order to prevent the DoS attack, Ethereum introduces a metering mechanism called gas. As the EVM executes a smart contract, it carefully accounts for every instruction (computation power, memory, etc.). Each instruction has a predetermined cost in units of gas. When a transaction triggers the execution of a smart contract, it must include an amount of gas that sets the upper limit of what can be consumed running the smart contract. The EVM will terminate execution if the amount of gas consumed by computation exceeds the gas available in the transaction. Gas is the mechanism Ethereum uses to allow Turing-complete computation while limiting the resources that any program can consume, while also resolving the halting problem.

The gas to pay for computation on the Ethereum network is bought with ether. So, ether is sent along with a transaction that is to be executed by the network which also contains the gas price allowed for each transaction. The ether sent is used to purchase gas required to perform the transaction and any unused gas post the execution of the transaction is refunded back to the sender of the transaction.

2.4 QTUM blockchain

The Qtum blockchain is a UTXO based smart contract system with a proof-of-stake consensus model. It uses the best of both Bitcoin and Ethereum blockchains. It has adopted the UTXO model of bitcoin which is more secure and uses the Ethereum Virtual Machine as its platform for contract execution. Qtum uses an Account Abstraction Layer (AAL) which maps the UTXO-based model to an account-based structure that is present in the EVM and achieves interoperability. It implements an on-chain governance system based on the Decentralised Governance Protocol (DGP) that allows QTUM token holders to participate in the voting and negotiation of the

upgrade and iteration of the blockchain network. It also introduces a way for other participants in the ecosystem, including developers, community member representatives, miners, and other multi-party participants to propose and vote for proposals. DGP manages the parameters responsible for the proper functioning of the blockchain network through smart contracts embedded within the genesis blocks.

Some of the problems that the Qtum blockchain addresses are:

1. Different blockchain platforms that exist today are not compatible with each other. For example, the Bitcoin ecosystem based on the UTXO (Unspent Transaction Output) model is not compatible with the Ethereum ecosystem based on the Account model, and the level of interoperability between these blockchains is not sufficient enough for businesses to adopt them.

2. On-chain governance of critical technical parameters that affect the system is difficult to achieve. For most decentralized platforms, once the mainnet deployment is completed, upgrade and governance of the blockchain is a major problem. Updates to Ethereum involved hard forks which are not backward compatible.

3. The Proof-of-Work consensus mechanism requires spending of large amount of energy for mining and incentives for miners. The security of the system is largely dependent on the hashing power of miners in the network, where there is a risk of centralization in mining computing power and thus the network faces the danger of a '51 percent' attack. [21]

2.4.1 Account Abstraction Layer (AAL)

The EVM is stack-based with a 256-bit machine word. Smart contracts that run on Ethereum use this virtual machine for their execution. The EVM is designed for the blockchain of Ethereum and thus, assumes that all value transfers use an account-based method. Qtum is based on the blockchain design of Bitcoin and uses the UTXO-based model. To translate the UTXO-based model to an account-based interface for the EVM and decouple the value transfer layer from the contract execution layer, Qtum created the Account Abstraction Layer (AAL). This facilitates interoperability and platform independence.

Qtum developed mechanisms for conversion between smart contract operations and UTXO operations, and has designed and developed four new opcodes:

- OP_CREATE: create a smart contract
- OP_CALL: call smart contract (send QTUM to the contract)
- OP_SPEND: spend QTUM in smart contract
- OP_SENDER: allow an address other than contract caller to pay for Gas

OP_CREATE passes the contract bytecode to the virtual machine. OP_CALL sends data, gasPrice, gasLimit, VMversion and other key parameters required to run smart contracts through transaction scripts, and finally passes them to the virtual machine. During the block creation process, the validator's script parses the contents in the transaction. In addition to making regular checks on transaction scripts, it also checks whether transactions contain the above-mentioned opcodes. Note that the above-mentioned opcodes are relate to operations that use funds which are stored as UTXOs. When the mentioned opcodes are encountered, those transactions are set aside to be separately processed. The contract transactions are then processed by the EVM into a special "Expected Contract Transaction List" which is executed by validator nodes. These transactions are then run on the EVM, and the resulting output is converted into a spendable Qtum transactions that support the operations related to EVM execution. Relying on this design, the Qtum x86 virtual machine can run on the blockchain in parallel with the EVM (Ethereum Virtual Machine), without the need to significantly modify the underlying protocol and retaining good functional scalability. Thus, any virtual machine based on the account model can be adapted to run on the Qtum blockchain when required.

Qtum has also adopted the concept of Gas from Ethereum which is a metering mechanism to track and restrict the use of resources belonging to the network. Use of the Gas model can prevent endless loops caused by errors and malicious attacks. They also encourage contract designers and users to make reasonable use of on-chain resources. Normally the address of the contract call sender pays the Gas, but the OP_SENDER opcode allows a third-party address, such as a distributed application service provider, to pay the Gas. As in Ethereum blockchain, there is also a state rollback for an 'out of Gas' scenario and a refund of remaining Gas after successful execution. [20]

2.4.2 Proof-of-Stake Algorithm

Proof of stake (PoS) is another type of consensus algorithm by which a cryptocurrency blockchain network aims to achieve distributed consensus. In PoS-based cryptocurrencies, the creator of the next block is chosen using various combinations of random selection and wealth or age. Here wealth and age refer to the stake of the validator who is responsible for the block creation and receives the transaction fees. A validator is the one who proposes the next block to be appended to the blockchain. The chance of a validator being selected is proportional to the total amount they have staked. In contrast, the algorithm of proof-of-work based blockchains such as bitcoin uses mining that comprises of solving computationally intensive puzzles to validate transactions and create new blocks.

Proof of stake must have a way of defining the next valid block in any blockchain. Selection by account balance would result in centralization, as the single richest member would have a permanent advantage. Instead, several different methods of selection have been devised. Certain blockchains that implement Proof-of-Stake algorithm use randomization to predict the validator of the next block, by using a formula that looks for the lowest hash value in combination with the size of the stake. Since the stakes are public, each node can predict with reasonable accuracy which account will next win the right to forge a block. [22]

Coin Aging

Some blockchains use a proof-of-stake system that combines randomization with the concept of ‘coin age’, a number derived from the product of the number of coins multiplied by the number of days the coins have been held by a validator. Older and larger sets of coins have a greater probability of signing the next block. For example, coins that have been unspent for at least 30 days begin competing for the next block. Once a stake of coins has been used to sign a block, its age is reset to zero and thus waits for at least 30 more days before signing another block. Also, the probability of finding the next block reaches a maximum after 90 days in order to prevent very old and very large collections of stakes from attaining a monopoly in the proof-of-stake consensus mechanism.

2.5 Smart Contracts and DApps

Until now, we have understood the concept of blockchain, different blockchains such as Bitcoin and Ethereum, Turing complete blockchains that run as Universal Turing Machines (UTM), which can execute code of unbounded complexity. The purpose of Turing complete blockchains is to execute immutable code (or programs) that run deterministically. This concept of immutable programs that execute in a deterministic fashion, facilitated by a Turing complete blockchain is referred to as a smart contract. A decentralized application or a DApp is more complex and is composed of at least two components namely, a smart contract on a blockchain and a web user interface that interacts with the contract. A DApp may also include other components such as, a decentralized storage protocol such as InterPlanetary File System (IPFS) and a decentralized messaging protocol (for example, whisper is a peer-to-peer messaging protocol).

2.5.1 Contract Components

Smart contracts are written in a high-level language such as solidity. There are different components in a contract that have to be properly understood in order to efficiently design, develop, deploy and call a contract. We have used solidity to develop smart contracts for the Ethereum blockchain. The different components in a smart contract are as follows:

- **Contract Bytecode**

Once the contract has been written they are compiled to get the bytecode. This contains a set of instructions that are to be executed by the Ethereum Virtual Machine (EVM). They are not human readable and is understood and used by the EVM.

- **Contract Address**

Once compiled, the bytecode is deployed on the Ethereum (or any other blockchain) network using a contract creation transaction. The newly created contract is identified by an Ethereum address, an outcome of the creation transaction. The address can be used in a transaction as a recipient in order to send funds to the transaction or to call one of its functions.

- **Application Binary Interface (ABI)**

An ABI is an interface between two program modules and defines how data and functions within the smart contract (bytecode) are accessed. It is the primary way to encode and decode data present as machine code on the blockchain. Ethereum uses an ABI to encode contract calls and read data from the contract by defining functions that can be invoked and their arguments. More specifically, an ABI is specified as a JSON array of data and functions defined within the contract.

- **Contract State**

The state of the contract refers to the data stored within the contract, is changed by the execution of transactions. It is only when the transaction executes successfully that the transactions are recorded globally. If an execution fails because of an error all of the changes made to the state is 'rolled back'. A failed execution is still recorded on the blockchain as an attempt.

2.5.2 Contract Development

Solidity is the high-level programming language we will use to develop smart contracts and understand their working. Solc is the compiler that converts high-level code written in Solidity to EVM bytecode. We will use a web-based development environment called Remix IDE for implementing smart contracts.

Let us consider a simple contract 'LateFlightReimbursement.sol' that makes payment for claims made, when flights are delayed.

```
pragma solidity ^0.4.25;
contract LateFlightReimbursement{
   uint256 public scheduledDepartureTime;
   uint256 public actualDepartureTime;
   bool public flag;
   address owner;
   address  insured;
   uint256 public ownerBalance;
   constructor(address wallet) public payable {
      flag=false;
      owner=msg.sender;
      ownerBalance=owner.balance;
```

```
    insured=wallet;
}
function setScheduledDepartureTime(uint256 _scheduledDepartureTime) public {
    scheduledDepartureTime=_scheduledDepartureTime;
}
function setActualDepartureTime(uint256 _actualDepartureTime) public payable{
    actualDepartureTime=_actualDepartureTime;
    if(actualDepartureTime>scheduledDepartureTime){
        insured.transfer(msg.value);
        flag=true;
        ownerBalance=owner.balance;
    }
    else
      flag = false;
}}
```

In the above example we observe that a smart contract is defined in solidity using the '*contract*' keyword. The contract consists of persistent variables (each of which consume space in the blockchain), a constructor that initializes the variables and functions. A function should be annotated with the '*payable*' keyword, in order that it may collect or receive funds in ether. The above contract accepts an 'address' value that represents the insured's wallet, which is defined in its constructor. It contains functions of setting the values of 'scheduled departure time' and 'actual departure time'. When the value of 'actual departure time' is greater than 'scheduled departure time' the function setActualDepartureTime() transfers ether to the insured's wallet using the *transfer()* method. The different data types used in the contract are *uint 256* represents unsigned int, *bool* defines a Boolean value and *address* data type is used to store and Ethereum address.

2.5.3 Contract Deployment

Before a contract is deployed, it is be compiled using a compiler. We use 'solc' to compile solidity code. When the code compiled using the solidity compiler, we get the ABI and bytecode that are important components that are used to deploy and call

the contract. Bytecode is the machine code that is deployed in the EVM and the ABI, which represents the structure of the contract is used to access the contract deployed in the blockchain.

The ABI for the above-mentioned contract is –

[

{ "constant": false, "inputs": [{"name": "_actualDepartureTime", "type": "uint256"}], "name": "setActualDepartureTime", "outputs": [], "payable": true, "stateMutability": "payable", "type": "function"},

{"constant": false, "inputs": [{"name": "_scheduledDepartureTime", "type": "uint256"}], "name": "setScheduledDepartureTime", "outputs": [], "payable": false, "stateMutability": "nonpayable", "type": "function"}, ……. ……. {"constant": true, "inputs": [], "name": "scheduledDepartureTime", "outputs": [{"name": "", "type": "uint256"}], "payable": false, "stateMutability": "view", "type": "function"}

]

Once compiled, the code is deployed onto the blockchain. In our case we have compiled and deployed the contract using Remix IDE (see figure 7). When the contract is deployed a transaction is executed on the blockchain, and a contract is created. The contract represents an account that is identified using an address obtained as the result of the contract creation transaction. After deployment, the components used to access the contract are the contract address and the ABI. An example contract address is ‘0xd4fe68a6aBC4e8BD59b9fB15f13c2A40cD883EE6’

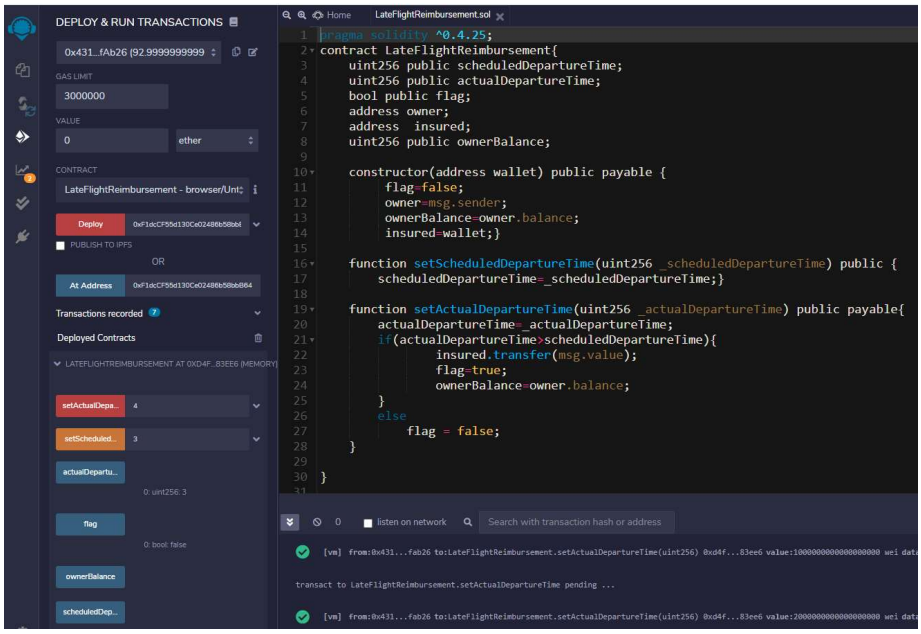

Figure 7: Deploying a contract using Remix IDE

Chapter 3

Oracles in Blockchain

In the previous section we considered an example smart contract, which contained functions that could be called from an externally owned account (EOA) to alter the state of the contract. The real-world use cases of smart contracts essentially have monetary value attached to them and vulnerabilities in the system can be exploited for financial gain. It is important that calls made to the contract, and data triggering certain actions such as transfer of funds within the contract is verified. Otherwise, an adversary could send incorrect data to the contract, resulting in the execution of code that is favourable to him/her. It can also happen that data is not made available to the smart contract when required and the contract conditions remain unexecuted. Hence it is important that correct data is made available to contracts on the blockchain and is done at the right time. This piece of work to be performed, is critical to the working of smart contracts on the block chain and is implemented using an Oracle.

3.1 Introduction to Oracles

An Oracle is essentially a system that answers a question that is external to the blockchain. It acts as a bridge that connects blockchains and the outside world. An oracle system consists of off-chain and on-chain components, where off-chain components are responsible for collecting the data in a secure fashion and on-chain components comprise of oracle's contracts that are used to send data to the contract that has requested for the oracle's service. They send extrinsic information to contracts on the blockchain by executing transactions that contain the data payload. However, such data simply cannot be trusted and used to execute high value contracts. It is important to create a trustworthy model of an oracle, so that they are acceptable by businesses to execute their use cases.

All oracles in general, contain a few critical functions by definition, which include the following –

• Collect data from a source external to the blockchain.
• Transfer that data on-chain together with signature.
• Data is made available to a smart contract's storage.

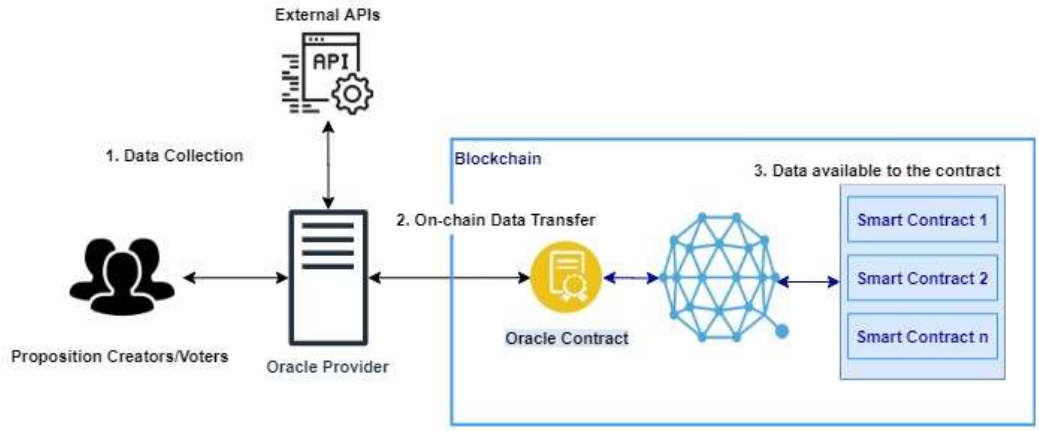

Figure 8: An Oracle System in general

It is important to note that oracles are not data sources, but a middleware that queries, verifies and authenticates data sources external to the blockchain and then passes the information to the smart contract on the blockchain that requested the information. Another key advantage of using an oracle is that it provides for off-chain computation. Computation and usage of resources on the blockchain is costly, and an

oracle can be used to perform large computations outside the network, with only the result being returned for use to the blockchain. There are many oracles that differ in their design and functionalities, currently in the market. How an oracle functions is dependent on the use case or purpose it is designed for.

Oracle Design Patterns

An oracle can be set-up in 3 three different known designs namely - immediate-read, publish-subscribe, and request–response. Oracles having immediate-read design is implemented in situations where data is required to make an immediate decision. For example, 'Is a person above 18 years of age?'. This type of an oracle stores data only once (which may be updated later) and other contracts on the blockchain that require that information can access it using a request call. Publish-subscribe setup is where an oracle broadcasts data to the network and those who have subscribed listen to the broadcasted data.

The request-response model is used when the data space required is too huge to be made available in a smart contract and users need only a small portion of the data for computation at a time. When an externally owned account interacts with a contract in the blockchain that uses the oracle, it results in an interaction with the oracle's contract which is essentially a request to the oracle, with the associated arguments detailing the data requested. The oracle then interacts with its contract to fetch the query and the different parameter, which is used to perform the actual query of the off-chain data source. The result of the query is signed by the oracle owner and delivered in a transaction to the contract that made the request, either directly or via the oracle's contract. [19]

Although blockchains can be setup in three different ways, oracles are fundamentally classified as centralized and decentralized. A centralized oracle is controlled by a single entity, whereas a decentralized oracle is a system where data is collected from several sources and controlled by one or more entities that are equally privileged. (See sections 3.2 and 3.3)

3.2 Centralized Oracles

A centralized oracle is governed by a single entity, which is the only provider of information for the smart contract. Having only one interface can be very risky, in the sense, a reliability of a contract that is secure on the blockchain is entirely dependent on the entity controlling the oracle. A bad actor can manipulate the data being sent from the oracle to the on-chain contract, which will have a direct impact on the smart contract. Thus, the main problem that exists with centralized oracles is of a single point of failure, which makes the contracts vulnerable to attacks. Contracts of higher value act as incentives for bad actors to attack the oracle system. One of the most widely used centralized oracle is Oraclize.

Oraclize (now is Provable)

Provable is the leading oracle provider in the blockchain industry, fulfilling thousands of requests every day on various platforms such as Ethereum, Hyperledger and R3 Corda. As explained already, an oracle is and intermediary service that is introduced as a result of blockchain contracts not having the ability to make request to external services that provide data which is critical to their functioning. But, to rely on an intermediary system, would be to betray the reduced-trust and security model of the blockchain. Therefore, it is important that a reliable and trustworthy model which does not comprise the security provided by the blockchain.

Provable follows a request-response model together with a concept of polling where requests are made until there is no more gas in the transaction that executes the provable query. It has developed a trust model which demonstrates that the data fetched from a source is not altered and is genuine. For this purpose, it sends data to the contract in the blockchain, together with an authenticity proof. Authenticity proofs are cryptographic guarantees that the data has not been tampered with. Provable uses TLSNotary proofs that allow it to provide an evidence that HTTPS web traffic occurred between the client and the server (data source provided by the contract). TLSNotary makes use of the TLS (Transport Layer Security) protocol to sign the data that has been accessed using the TLS master key. The signed data is split between three parties – the server which is the oracle, an auditee (the oraclize service) and an auditor. The provable service uses the AWS machine instance as the auditor which

verifies that the data has not been modified since instantiation. The TLSNotary secret is stored safely in the instance and is used to provide honesty proofs.

A valid request from a smart contract to the provable service is of the format –

- A data source, such as a URL to an API
- A query
- An authenticity proof (optional)

Data Sources

A data source is a provable query refers to a data source that the contract delegates as its trusted source. E.g., nasdaq.com. Provable currently offer services for data sources such as a URL, Wolfram Alpha, IPFS, Random and Computation. AURL can be an API or a web page. Wolfram Alpha is a computational engine that can be specified as a data source in the provable query to retrieve an answer to the query string provided. Random is used to generate an untampered random value.

Query

A query is an array of parameters, which is used by the oraclize service to process the request. For example, in the case of the data source being specified as ‘Wolfram Alpha’, the query parameter can be ‘Flip a coin’ which generates a random value which is either a ‘heads’ or a ‘tails’.

Authenticity Proofs

Authenticity proof form a critical part of trust model of Provable. They refer to the type of authenticity proofs required to prove that the data supplied has not been tampered with. E.g., TLSNotary and proof storage IPFS .

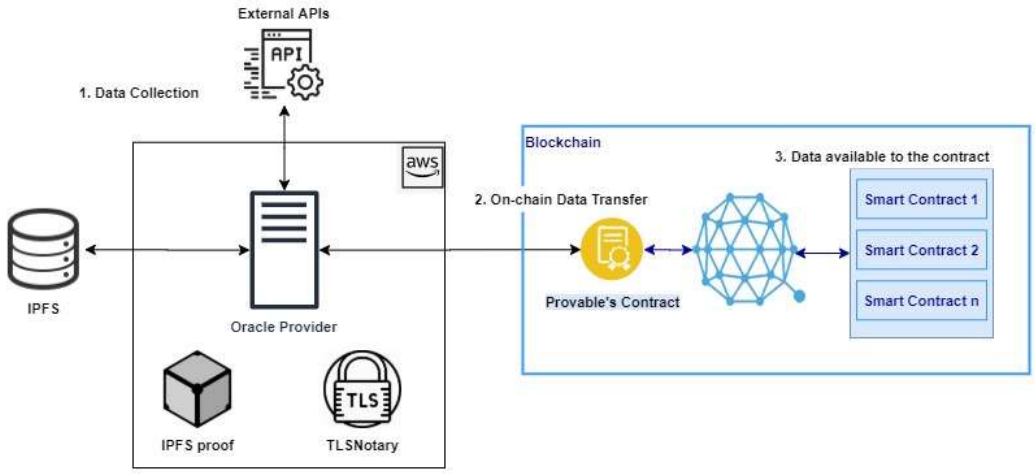

Figure 9: Provable - A Centralized Oracle System

3.3 Decentralized Oracles

Bibliography

[1] Kolb, J., Abdelbaky, M., Katz, R. H., & Culler, D. E. (2020). Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial. ACM Computing Surveys, 53(1), 1–39. https://doi-org.elib.tcd.ie/10.1145/3366370

[2] Satoshi Nakamoto. (2009). Bitcoin: A Peer–to–Peer Electronic Cash System. Retrieved from: https://bitcoin.org/bitcoin.pdf .

[3] Vitalik Buterin. (2014). A Next-Generation Smart Contract and Decentralized Application Platform. Retrieved from: https://github.com/ethereum/wiki/wiki/White-Paper .

[4] Feng, T., Yu, X., Chai, Y. and Liu, Y. (2019). Smart contract model for complex reality transaction. International Journal of Crowd Science, Vol. 3, pp. 184-197.

[5] M. Merlini, N. Veira, R. Berryhill and A. Veneris. (2019). On Public Decentralized Ledger Oracles via a Paired-Question Protocol. IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Seoul, Korea (South), pp. 337-344.

[6] Oraclize.it. [Online]. Available: http://www.oraclize.it/

[7] Karame, G.O., & Audroulaki, E. (2016). Bitcoin and Blockchain Security.

[8] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in Proceedings of the 2016 aCM sIGSAC conference on computer and communications security. ACM, 2016, pp. 270–282.

[9] V. Costan and S. Devadas, "Intel sgx explained." IACR Cryptology ePrint Archive, vol. 2016, no. 086, pp. 1–118, 2016.

[10] Berberich, M.; Steiner, M. (2016). Blockchain technology and the gdpr how to reconcile privacy and distributed ledgers. European Data Protection Law Review (EDPL), 2(3), 422-426.

[11] Blockchain Oracles Explained. Available: https://www.binance.vision/blockchain/blockchain-oracles-explained

[12] Bitcoin's UTXO Set Explained. Available: https://www.mycryptopedia.com/bitcoin-utxo-unspent-transaction-output-set-explained/

[13] Full Node. Available: https://en.bitcoin.it/wiki/Full_node

[14] Unspent transaction output. Available: https://en.wikipedia.org/wiki/Unspent_transaction_output

[15] Domain Name System. Available: https://en.wikipedia.org/wiki/Domain_Name_System

[16] Remote Procedure Call. Available: https://en.wikipedia.org/wiki/Remote_procedure_call

[17] Dr. Gavin Wood. (2019). Ethereum: A secure decentralised generalised transaction ledger byzantium version. Retrieved from: https://ethereum.github.io/yellowpaper/paper.pdf

[18] Vitalik Buterin. (2013). Ethereum Whitepaper. Retrieved from: https://ethereum.org/whitepaper/

[19] Antonopoulos, A. M., & Wood, G. (2018). Mastering Ethereum: Building smart contracts and DApps.

[20] Patrick Dai, Neil Mahi, Jordan Earls, Alex Norta. Smart-Contract Value-Transfer Protocols on a Distributed Mobile Application Platform. Retrieved from: https://old.qtum.org/en/white-papers

[21] QTUM Blockchain New Whitepaper. (2020). Retrieved from: https://qtum.org/en/developer

[22] Proof of stake. Retrieved from: https://en.wikipedia.org/wiki/Proof_of_stake

[23] Sin Kuang Lo, Xiwei Xu, Mark Staples, Lina Yao. (2020). Reliability analysis for blockchain oracles. Volume 83.

[24] Abdeljalil Beniiche. (2020). A Study of Blockchain Oracles. Retrieved from: https://deepai.org/publication/a-study-of-blockchain-oracles

Innovation:

Designated source (considered a proposition) and later voting and certification takes place.