**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 Master 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 at 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [Contents](#page8) | | | | viii |
| [List of Figures](#page10) | | | | x |
| [List of Tables](#page12) | | | | xii |
| [1](#page13) | [Introduction](#page13) | | | 1 |
|  | [1.1](#page13) | Motivation | |  |
|  | [1.2](#page15) | Research Question | |  |
|  | [1.3](#page16) | Research Objective | |  |
|  | [1.4](#page17) | Research Challenges | |  |
|  | [1.5](#page18) | Thesis Overview | |  |
|  | [1.6](#page18) | Thesis Structure | |  |
| [2](#page20) | Literature Review | | |  |
|  | 2.1 | Blockchain | |  |
|  |  | 2.1.1 | Blockchain as a Ledger |  |
|  | [2.2](#page20) | Bit-coin | |  |
|  |  | 2.2.1 | UTXO model |  |
|  |  | 2.2.2 | Proof-of-Work algorithm |  |
|  |  | 2.2.3 | Merkel Trees |  |
|  | [2.3](#page21) | Ethereum | |  |
|  |  | 2.3.1 | Accounts in Ethereum |  |
|  |  | 2.3.2 | Messages and Transactions |  |
|  |  | 2.3.3 | Turing Completeness in Ethereum |  |
|  | 2.4 | QTUM blockchain | |  |
|  |  | 2.4.1 | Account Abstraction Layer |  |
|  |  | 2.4.2 | Proof-of-Stake algorithm |  |
|  | [2.5](#page28) | Smart Contracts and DApps | |  |
| [3](#page43) Oracles in Blockchain | | | |  |
|  | [3.1](#page43) | Introduction to Oracles | |  |
|  | [3.2](#page45) | Centralized Oracles | |  |
|  | [3.3](#page47) | Decentralized Oracles | |  |
|  | [3.4](#page48) | Use Cases | |  |
| [4](#page50) | [Design](#page50) | | |  |
|  | [4.1](#page50) |  | |  |
|  | [4.2](#page51) |  | |  |
|  | [4.3](#page59) |  | |  |
|  | [4.4](#page60) |  | |  |
| [5](#page61) | [Implementation](#page61) | | |  |
|  | [5.1](#page61) |  | |  |
|  | [5.2](#page67) |  | |  |
|  | [5.3](#page70) |  | |  |
|  | [5.4](#page75) |  | |  |
| [6](#page77) | [Evaluation](#page77) | | |  |
|  | [6.1](#page77) |  | |  |
|  | [6.2](#page78) |  | |  |
|  | [6.3](#page90) |  | |  |
| [7](#page94) | [Conclusion](#page94) | | |  |
|  | [7.1](#page94) |  | |  |
|  | [7.2](#page96) |  | |  |
| [Bibliography](#page99) | | | |  |

Chapter 1

Introduction

Blockchain is a technology that is used to reach an agreement on the order of events among a set of entities. It is a dynamic area of research, investment and product development. The reason for this enthusiasm is that it allows for the implementation of a decentralised system. There is no single entity that fully controls the operation in the blockchain. Instead, it uses cryptography and incentive schemes that are carefully designed to help create an immutable and secure distributed ledger.

By design, the blockchain network has no central authority with the ability to unilaterally approve invalid transactions, roll back previous transactions, or otherwise manipulate the state of the system through any means aside from normal submission of transactions for processing. The distributed ledger used is an append-only log for storing data. A new transaction can be added only at the end of the ledger, and no previous transactions can be modified. Each transaction is signed using the private key of the user who initiates the transaction, preventing forgery. Moreover, a new transaction is not immediately added to the ledger but is bundled together with other pending transactions into a block. [1]

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 by automating the decision-making process.

* 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 Alice and Bob place a bet on who the winner of the US presidential election will be. Alice believes that the Republican candidate will win, while Bob 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 cannot interact with external data, it must depend on an oracle for the necessary information – in this case, the results of the presidential election. After the election is over, the oracle queries a trusted API 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. Without the oracle relaying the data, there would have been no way to settle this bet in a way that couldn’t be gamed by one of the participants. [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 malicious operating systems.

Oracles are essential implementations in numerous applications where there is a need for data that is external to be brought onto 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. Thus, the oracles in the Blockchain eco-system remain a subject of research and innovation. In this work we try to 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 is an innovation of the existing architecture.

* 1. **Research Question**

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

* 1. **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. **Challenges**
7. Lack of substantial literature on Oracles and their implementations.
8. Bringing innovation to the existing work on Oracles.
9. Lack of proper conviction in the Blockchain system.
10. Insufficient documentation for oraclize.it service.
11. Few implementations of smart contracts and oracles available on the web or other resources.
12. Compatibility of solidity compilers was often an issue. Hence had to avoid using compilers having versions greater than 4.25.
    1. **Overview**

Using oracles to send data that is external into the blockchain that is useful for the decentralized applications deployed is an interesting development in the blockchain industry. In order to understand 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. It discusses the various oracles currently available in the market and categorises them as centralized and decentralized. We also 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. 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. **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

* 1. **Blockchain**

Modern day payment systems facilitate exchange of money between two entities—a payer and a payee. Apart from the payer and payee, a payment system traditionally involves two more entities; one entity that manages assets and/or 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 acquiring bank or acquirer.

The operations of a typical cash-like system are depicted in Figure 2.1. In a cash-like system, the payer’s account is charged before the actual payment that takes place. Transactions require that a payment be made to the intermediary bank which usually is a percentage of the amount that is transferred. Thus, international enterprises can face high banking fees when the total amount transferred and the number of transactions increase. [7] Further, the amount is transferred through intermediaries that maintain their own logs, which could be altered by corrupt parties. Blockchain can improve the payment system by offering added security, higher transfer speeds and lower conversion fees. 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.

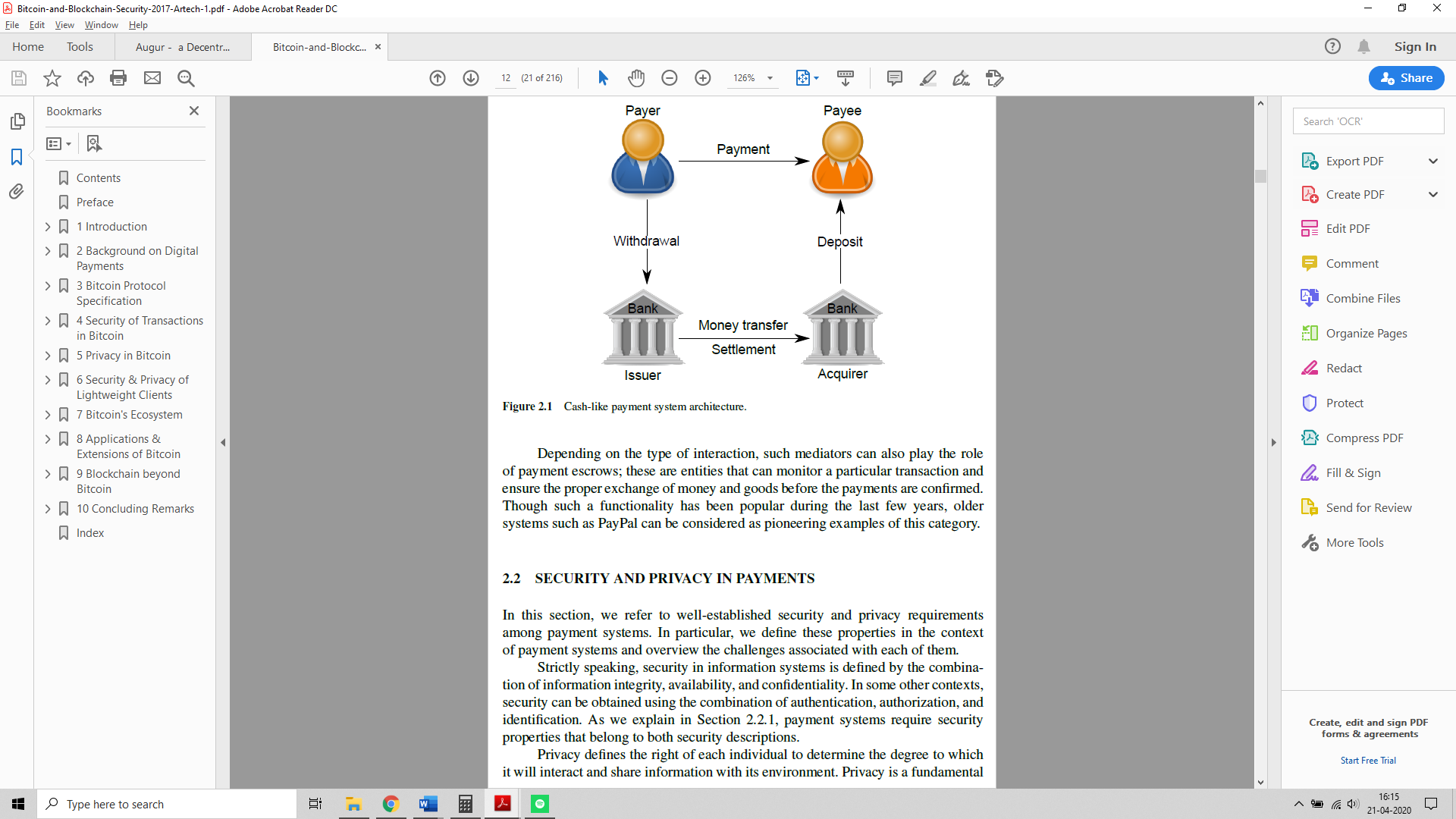


Figure 1: Cash-like Payment System Architecture

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]

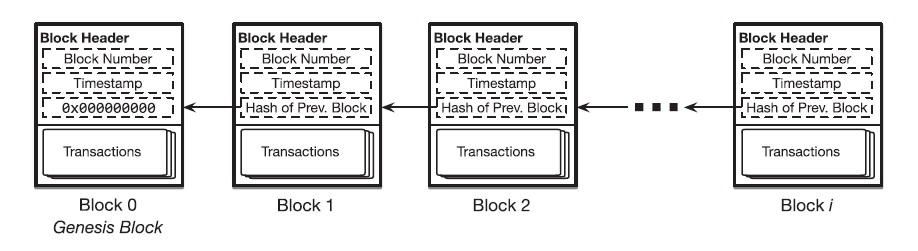


Figure 2: A Simple Blockchain

* + 1. **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, stored on every node of the system as a complete copy of the blockchain. 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.

* 1. **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 in the sequel, from the payer to the payee. These entities are called “peers,” and are referenced in transactions by Bitcoin addresses. Each address maps to a unique public/private key pair; these keys are used to transfer the ownership of BTCs among addresses. A Bitcoin address is an identifier of 26 to 35 alphanumeric characters.

The bitcoin blockchain operates on top of a loosely connected P2P network, where nodes can join and leave the network at will. The peers 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 nodes connected to the network. For example, the domain name www.example.com translates to the addresses 93.184.216.34 (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 formed by digitally signing a hash of the previous transaction where this coin was last spent along with the public key of the future owner and incorporating this signature in the coin. Transactions take as input the reference to an output of another transaction that spends the same coins and output the list of addresses that can collect the transferred coins. A transaction output can only be redeemed once, after which the output is no longer available to other transactions. Once ready, the transaction is signed by the user and broadcast in the P2P network. Any peer can verify the authenticity of a BTC by checking the chain of signatures.

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) scheme (see section 2.2.2); more specifically, miners must find a nonce value that, when hashed with additional fields (e.g., 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. Due to the underlying PoW scheme, different miners can potentially find different blocks nearly at the same time—in which case a fork in the blockchain occurs. Forks are inherently resolved by the Bitcoin system; 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 therefore 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 Alice’s to be valid, it 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. 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.

**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. Conversely, when a new unspent output is created, 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 deleted when a transaction occurs, while at the same time, outputs are created in the form of UTXOs. These unspent transaction outputs may be used (by the holders of private keys; for example, persons with cryptocurrency wallets) for the purpose of 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 be used in for a future transaction. This allows bitcoins to move from one owner to another, with each transfer consuming and creating UTXOs in a chain of transactions. [12]

**Components in a Transaction Output**

scriptPubKey is a locking script placed on the output of a Bitcoin transaction that requires certain conditions to be met for a recipient to spend his/her bitcoins. Conversely, scriptSig is the unlocking script that satisfies the conditions placed on the output by the scriptPubKey, and thus, 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.

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 before he can 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 performed by the peers in the network. When users add a new entry to a blockchain’s ledger, they submit a transaction to an existing member 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. This means that the parties involved in a transaction, such as the sender and receiver of Bitcoins in a transaction on the Bitcoin blockchain, are not directly involved in the execution of that transaction. Instead, this task falls to the members of the network.

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 and by retaining a copy of the blockchain users do not have to trust an intermediary 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 is known as mining, and these members are therefore referred to as miners. Miners have to follow a certain protocol to ensure the property of consensus—all members of the blockchain’s network reach agreement about each new block to add to the chain and also 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, users may choose to run miners, because they have a vested interest in the successful operation of the blockchain or because of more explicit incentives (discussed below).

In Bitcoin and several other blockchains, miners follow a proof of work algorithm (Figure 2) 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 any 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. Instead, each newly appended block must also contain a random quantity, a nonce, such that a cryptographic hash of the block’s contents, including the nonce, falls below an upper threshold in its representation.

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 while 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 –

1. Accumulate a batch of pending transactions that have been received from the peers on the network but have not yet been included in any previous block and package 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. That is, 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, including n, to the network. Return to Step 1.

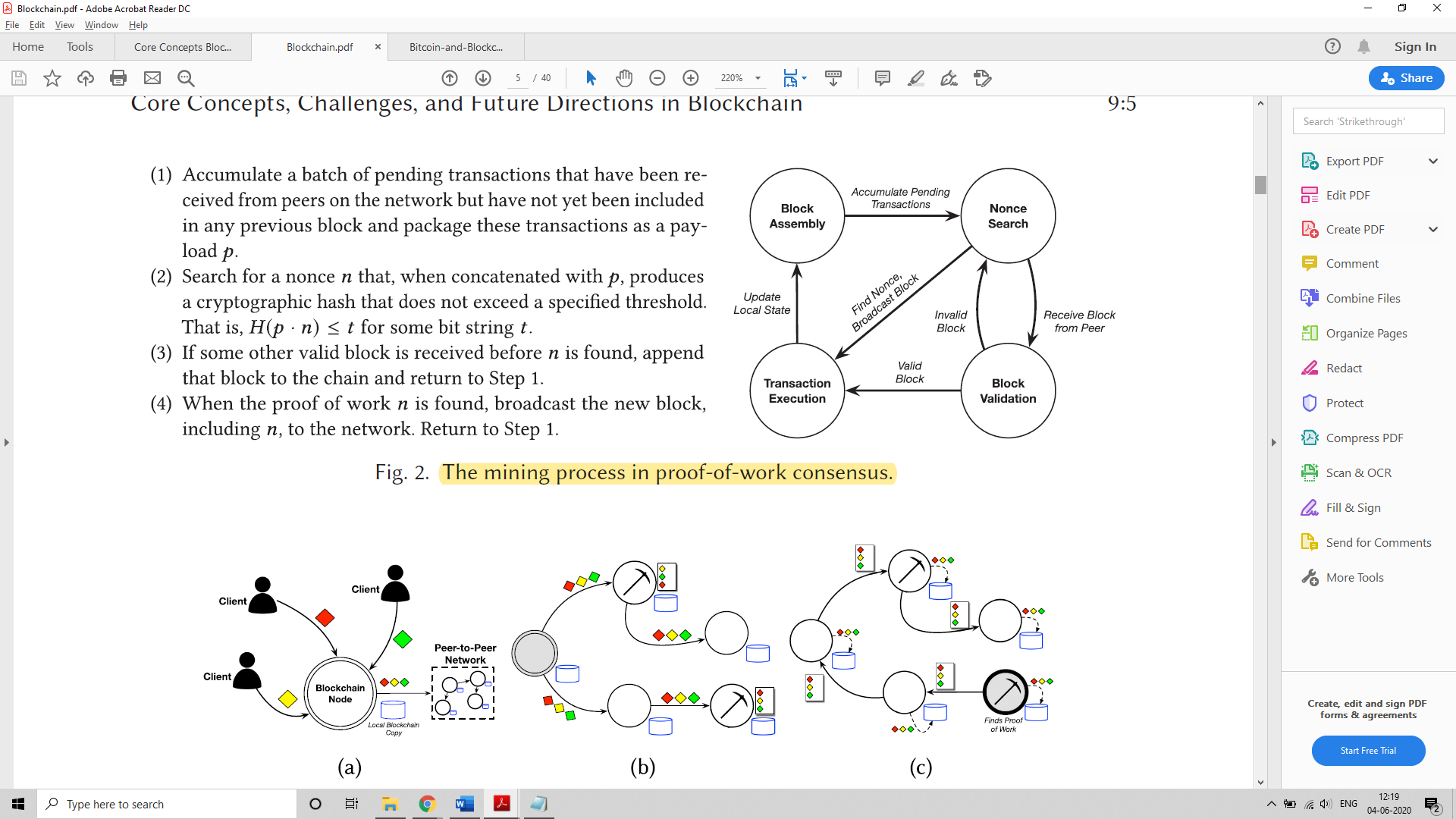


Figure 3: Proof-of-work consensus

Thus, proof-of-work can be considered a repeated lottery that determines which miner is allowed to dictate the next block in the chain. All other miners validate and accept the new block before moving on to the next round of the protocol and a new lottery. A particular miner’s odds of winning a lottery round (and thus 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 to computing power also gives proof-of-work consensus resilience to Sybil attacks – a technique in which a single adversary masquerades as many synthetic users of a system to gain control of that system.

The quantity t is a bit string representing an upper bound on the output of the cryptographic hash function used to produce a 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 nonces that can serve as a valid proof of work, while a larger t value increases the number of such nonces. Therefore, mining difficulty determines the expected number of nonces 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 (and thus the rate at which nonce values can be tested) fluctuates over time as nodes join and leave the network. [1]

**Forks**

Proof-of-work consensus is essentially leader election by lottery. The miner of each new block is chosen non-deterministically, which can lead to a split view of the blockchain’s state among its participants, known as a fork in the chain and depicted in Figure 4.

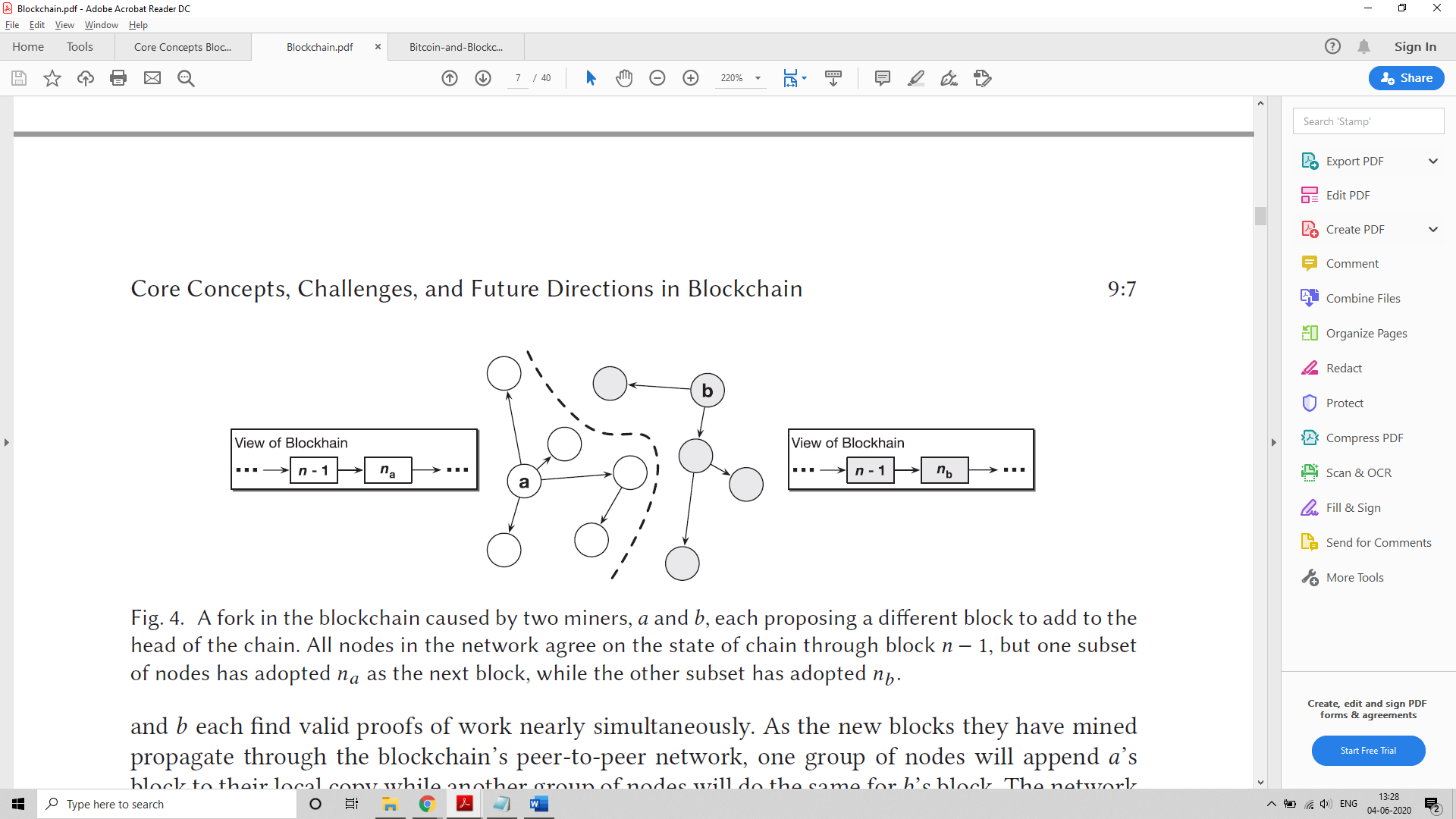


Figure 4: Fork in a blockchain

Imagine that two nodes a and b, each find valid proofs of work nearly simultaneously. As the new blocks they have mined propagate through the blockchain’s peer-to-peer network, one group of nodes will append a’s block to their local copy while another group of nodes will do the same for b’s block. The network is more vulnerable during a fork, because the hashing power of the network is now split into two competing groups, which introduces security vulnerabilities.

This is resolved by adding a simple rule to the proof-of-work consensus protocol: When a node observes a fork in the blockchain, it should always follow the longer of the two chains, i.e., the chain that contains more blocks. Every node should treat this longer fork as the canonical state of the chain, and every miner should add new blocks onto the head of this chain. One of the two forks will have its first block adopted by a subset of the network with more hashing power. This subset of the network will then be able to mine blocks at a faster rate than the nodes backing the second fork, so it will become longer with time, and this disparity will only grow as more nodes identify the longer fork and abandon the shorter fork.

The combination of the blockchain’s structure, where each block contains a hash of its predecessor, and the possibility of forks has led to the notion 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 take effect throughout the network. The block in question could turn 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. 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 confirmation blocks have been appended as successors on the chain to the transaction’s block. Each successor reduces the probability that the transaction is reverted by an attacker or discarded as part of an abandoned fork. [1]

**2.2.3 Merkel Trees**

Merkle trees allow the combination of multiple input sequences in a hash tree converging into the topmost Merkle root hash. This data structure allows the compact representation of a set of transactions, such as when the tree is built up from the transaction hashes (see Figure 5). Merkle trees can be used to instantiate cryptographic accumulators, which answer a query whether a given candidate belongs to a set.

A Merkle tree is a binary tree in which the data is stored in the leaves. More specifically, given a tree of height l, a Merkle tree accumulates elements of a set X by assigning these to the leaf nodes (starting from position 0). Let a[i, j] denote a node in the tree located at the ith level and jth position. Here, the level refers to the distance (in hops) to the leaf nodes; clearly, leaf nodes are located at distance 0. On the other hand, the position within a level is computed incrementally from left to right starting from position 0; for example, the leftmost node of level 1 is denoted by a[1, 0]. In a Merkle tree, the intermediate nodes are computed as the hash of their respective child nodes; namely a[i+1, j] = H(a[i, 2j], a[i, 2j+1]), where H(X) refers to the cryptographic hash of X. Figure 5 depicts an example of a Merkle tree accumulating eight elements. Here, a30 is referred to as the Merkle root and commits to all leaf elements U0, …, U7. To prove the membership of element U3 (highlighted in Figure 3.1) in the root a30, intermediate nodes a02, a10, and a21 (highlighted in ovals in Figure 3.1) are needed. We say that these nodes form the sibling path of U3. Given n leaves, Merkle trees require O(n) for constructing the tree and O(log(n)) to prove membership of any element in the tree.

Formally, a Merkle tree comprises the following algorithms:

D 🡨 Acc(X) – This algorithm accumulates the elements of a set X into a digest. Here, corresponds to the root node (i.e., D = a[l, 0]). This can be used to prove that the exact set X is correctly accumulated in D.

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 a[l, 0].

Verify m(D, x, Pm) – Given Pm, an element x, its sibling path and the root a[l, 0], this algorithm outputs true if and only if = a[l, 0] where l is the length of the sibling path and the sibling path of x matches the root a[l, 0]. [7]

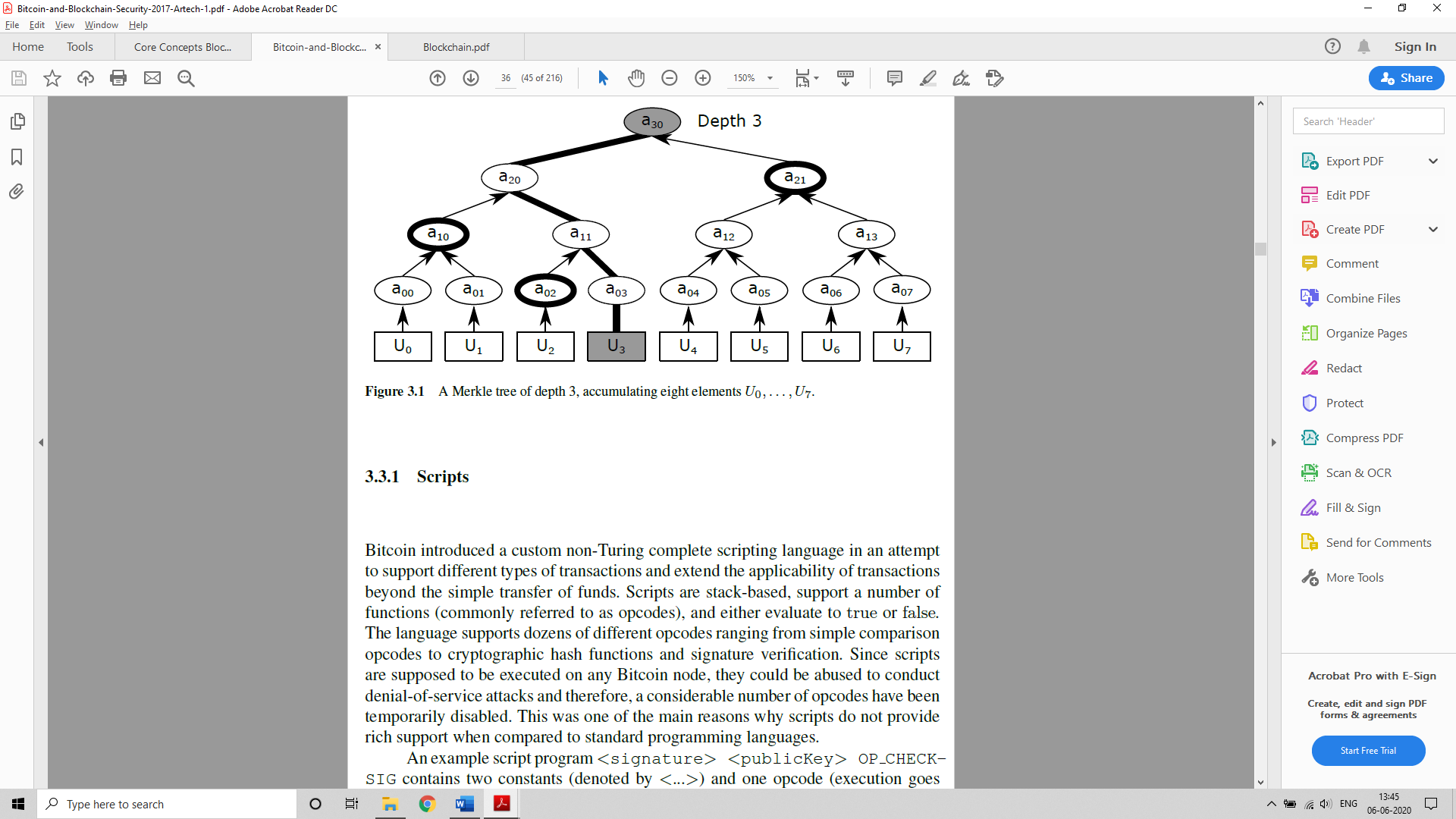


Figure 5: A Merkle tree of depth 3, accumulating eight elements U0, .... ,U7

An important scalability feature of Bitcoin is that the block is stored in a multi-level data structure. The "hash" of a block is actually only the hash of the block header, a roughly 200-byte piece of data that contains the timestamp, nonce, previous block hash and the root hash of a data structure called the Merkle tree storing all transactions in the block. A Merkle tree is a type of binary tree, composed of a set of nodes with a large number of leaf nodes at the bottom of the tree containing the underlying data, a set of intermediate nodes where each node is the hash of its two children, and finally a single root node, also formed from the hash of its two children, representing the "top" of the tree. The purpose of the Merkle tree is to allow the data in a block to be delivered piecemeal: a node can download only the header of a block from one source, the small part of the tree relevant to them from another source, and still be assured that all of the data is correct. The reason why this works is that hashes propagate upward: if a malicious user attempts to swap in a fake transaction into the bottom of a Merkle tree, this change will cause a change in the node above, and then a change in the node above that, finally changing the root of the tree and therefore the hash of the block, causing the protocol to register it as a completely different block (almost certainly with an invalid proof of work). [18]

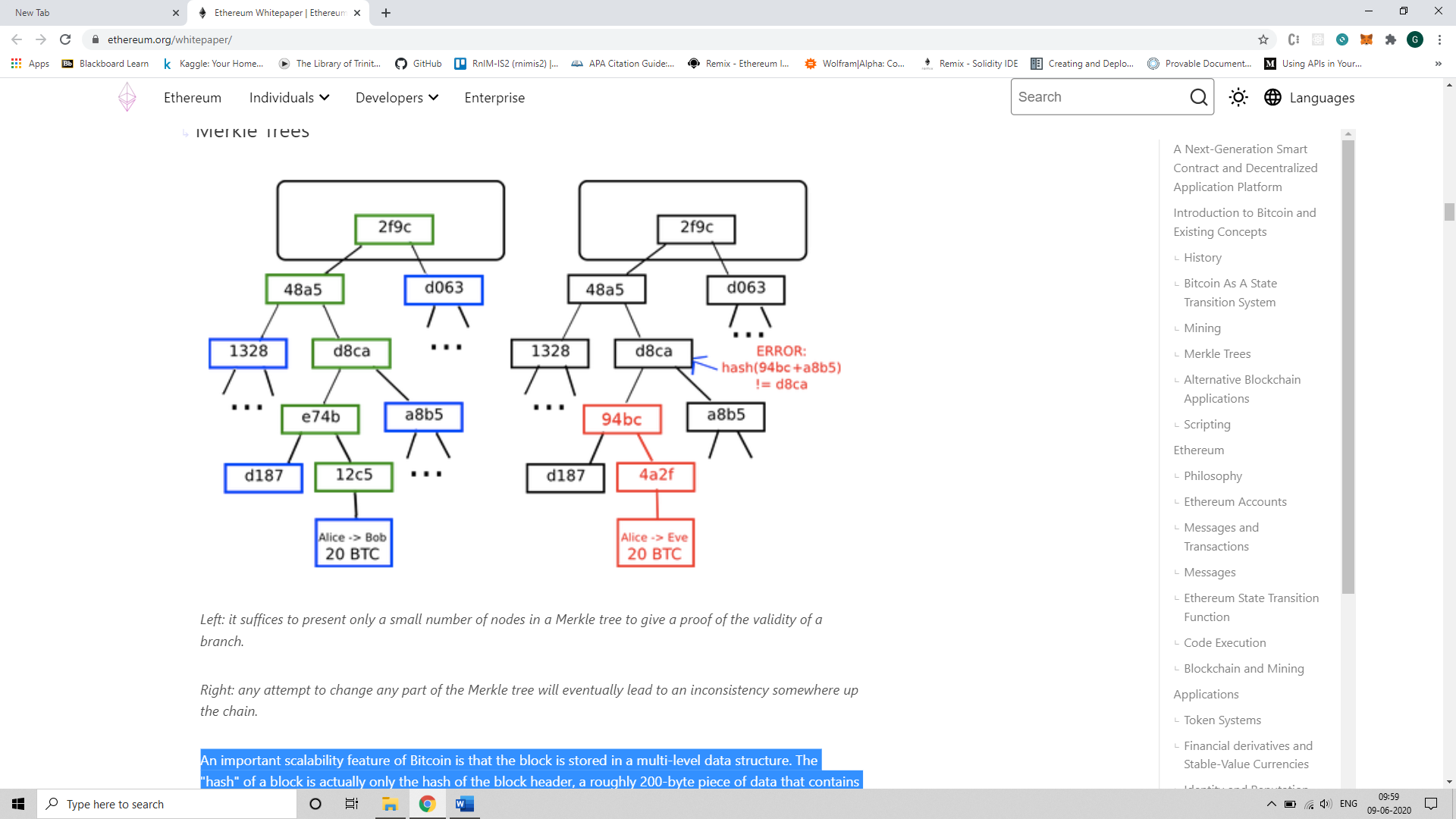


Figure 6: An attempt to change the Merkel Tree

* 1. **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 of state updates (a proof-of-work blockchain), the use of cryptographic primitives such as digital signatures and hashes, and a digital currency (ether). Yet in many ways the purpose and construction of Ethereum are strikingly 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 both integral to and necessary for the operation of Ethereum, ether is intended as a utility currency to pay for use of the Ethereum platform as a world computer.

Unlike Bitcoin, which has a very limited scripting language, Ethereum is designed to be a general-purpose programmable blockchain that runs a virtual machine capable of executing code of arbitrary and unbounded complexity. Where Bitcoin’s Script language is, intentionally, constrained to simple true/false evaluation of spending conditions, Ethereum’s language is Turing complete (see section 2.3.3), meaning that Ethereum can straightforwardly function as a general-purpose computer. The idea was that by using a general-purpose blockchain like Ethereum, a developer could program their particular application without having to implement the underlying mechanisms of peer-to-peer networks, blockchains, consensus algorithms, etc. The Ethereum platform was designed to abstract these details and provide a deterministic and secure programming environment for decentralized blockchain applications. Contrary to the bitcoin that tracks only the state of currency ownership Ethereum tracks the state transitions of a general-purpose data store, i.e., a store that can hold any data expressible as a key–value tuple. A key–value data store holds arbitrary values, each referenced by some key; for example, the value “Mastering Ethereum” referenced by the key “Book Title”. In some ways, this serves the same purpose as the data storage model of Random Access Memory (RAM) used by most general-purpose computers. Ethereum has memory that stores both code and data, and it uses the Ethereum blockchain to track how this memory changes over time. Like a general-purpose stored-program computer, Ethereum can load code into its state machine and run 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. [19]

* + 1. **Accounts in Ethereum**

Similar to the Unspent Transaction Output Model 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, if present
* The account's storage (empty by default)

"Ether" is the main internal 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, 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.

Note that "contracts" in Ethereum should not be seen as something that should be "fulfilled" or "complied with"; rather, they are more like "autonomous agents" that live inside of the Ethereum execution environment, always executing a specific piece of code when "poked" by a message or transaction, and having direct control over their own ether balance and their own key/value store to keep track of persistent variables. [18]

* + 1. **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. Transactions contain:

* The recipient of the message
* A signature identifying the sender
* The amount of ether to transfer from the sender to the recipient
* An optional data field
* A STARTGAS value, representing the maximum number of computational steps the transaction execution is allowed to take
* A GASPRICE value, representing the fee the sender pays per computational step

The first three are standard fields expected in any cryptocurrency. The data field has no function by default, but the virtual machine has an opcode which a contract can use to access the data; as an example use case, if a contract is functioning as an on-blockchain domain registration service, then it may wish to interpret the data being passed to it as containing two "fields", the first field being a domain to register and the second field being the IP address to register it to. The contract would read these values from the message data and appropriately place them in storage.

The STARTGAS and GASPRICE fields are crucial for Ethereum's anti-denial of service model. In order to prevent accidental or hostile infinite loops or other computational wastage in code, each transaction is required to set a limit to how many computational steps of code execution it can use. The fundamental unit of computation is "gas"; usually, a computational step costs 1 gas, but some operations cost higher amounts of gas because they are more computationally expensive, or increase the amount of data that must be stored as part of the state. There is also a fee of 5 gas for every byte in the transaction data. The intent of the fee system is to require an attacker to pay proportionately for every resource that they consume, including computation, bandwidth and storage; hence, any transaction that leads to the network consuming a greater amount of any of these resources must have a gas fee roughly proportional to the increment. [18]

**Messages**

Contracts have the ability to send "messages" to other contracts. Messages are virtual objects that are never serialized and exist only in the Ethereum execution environment. A message contains:

* The sender of the message (implicit)
* The recipient of the message
* The amount of ether to transfer alongside the message
* An optional data field
* A STARTGAS value

Essentially, a message is like a transaction, except it is produced by a contract and not an external actor. A message is produced when a contract currently executing code executes the CALL opcode, which produces and executes a message. Like a transaction, a message leads to the recipient account running its code. Thus, contracts can have relationships with other contracts in exactly the same way that external actors can.

Note that the gas allowance assigned by a transaction or contract applies to the total gas consumed by that transaction and all sub-executions. For example, if an external actor A sends a transaction to B with 1000 gas, and B consumes 600 gas before sending a message to C, and the internal execution of C consumes 300 gas before returning, then B can spend another 100 gas before running out of gas. [18]

* + 1. **Turing Completeness in Ethereum**

The term refers to 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 (resembling an infinite-length paper tape). With this construct, Turing went on to provide a mathematical foundation to answer (in the negative) 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 possible, given an arbitrary program and its input, to determine whether the program will eventually stop running. Ethereum’s ability to execute a stored program, in a state machine called the Ethereum Virtual Machine, while reading and writing data to memory makes it a Turing-complete system and therefore a UTM. Ethereum can compute any algorithm that can be computed by any Turing machine, given the limitations of finite memory.

Ethereum’s ground-breaking innovation is to combine the general-purpose computing architecture of a stored-program computer with a decentralized blockchain, thereby creating a distributed single-state (singleton) world computer. Ethereum programs run “everywhere,” yet produce a common state that is secured by the rules of consensus. 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. The fact that Ethereum is Turing complete means that any program of any complexity can be computed by Ethereum. But that flexibility brings some thorny security and resource management problems. An unresponsive printer can be turned off and turned back on again. That is not possible with a public blockchain.

**Implications**

Turing proved that you cannot predict whether a program will terminate by simulating it on a computer. In simple terms, we cannot predict the path of a program without running it. Turing-complete systems can run in “infinite loops,” a term used to describe a program that does not terminate. 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 between the starting conditions and the code. In Ethereum, this poses a challenge: every participating node (client) must validate every transaction, running any smart contracts it calls. But as Turing proved, Ethereum can’t predict if a smart contract will terminate, or how long it will run, without actually running it (possibly running forever). Whether by accident or on purpose, a smart contract can be created such that it runs forever when a node attempts to validate it. This is effectively a DoS attack. And of course, between a program that takes a millisecond to validate and one that runs forever are an infinite range of nasty, resource-hogging, memory-bloating, CPU-overheating programs that simply waste resources. In a world computer, a program that abuses resources gets to abuse the world’s resources. How does Ethereum constrain the resources used by a smart contract if it cannot predict resource use 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, data access, 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.

The gas to pay for computation on the Ethereum network is bought with ether. Ether needs to be sent along with a transaction and it needs to be explicitly earmarked for the purchase of gas, along with an acceptable gas price. Gas is purchased for the transaction, the computation is executed, and any unused gas is refunded back to the sender of the transaction.

* 1. **QTUM blockchain**

Qtum blockchain is a UTXO based smart contract system with a proof-of-stake consensus model. Qtum uses the Ethereum Virtual Machine (EVM) that is integrated with the UTXO model using 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 on-chain governance proposals. DGP manages the parameters of the blockchain network through smart contracts embedded in the genesis blocks and clarifies the governance seats and proportion of governance participants for each party.

Some of the problems that the Qtum blockchain addresses are:

1. There is insufficient compatibility between different blockchain platforms. 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 interoperability between blockchains is not strong.

2. On-chain governance of critical technical parameters is difficult to achieve. For most decentralized platforms, once the mainnet deployment is completed, upgrade and governance of the blockchain is a major problem.

3. The consensus mechanism lacks flexibility. The Proof-of-Work consensus mechanism has certain limitations in terms of energy requirements and incentives for miners and currency holders, and there is a risk of centralization in mining computing power that could lead to a 51 percent attack.

* + 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 optimizations for the interface and conversion between smart contract operations and UTXO operations, 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 address other than contract call sender to pay for Gas

When the Qtum blockchain generates new blocks, in addition to making regular checks on transaction scripts, it also needs to check whether transactions contain the above-mentioned opcodes. OP\_CREATE is used to pass 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. 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. In the future, any virtual machine based on the account model can be adapted to run on the Qtum blockchain.

Qtum also borrowed the concept of Gas from Ethereum, used the Gas model in the contract operation, and optimized the Gas model of the EVM. Use of the Gas model can prevent endless loops caused by errors and malicious attacks, can allow miners to get rewards for performing calculations based on actual workload, and 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. Similar to EVM, there is also a state rollback for “out of Gas” and a refund of remaining Gas after successful execution.

* + 1. **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. In contrast, the algorithm of proof-of-work-based cryptocurrencies 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 following generator 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.

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. 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, it must start over with zero "coin age" and thus wait 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 or very large collections of stakes from dominating the blockchain.

* 1. **Smart Contracts and DApps**

Mastering Ethereum page 10

Chapter 3

Oracles in Blockchain

A subclass of blockchain oracles are decentralized blockchain oracles. We will consider an oracle decentralized if it is both permissionless (any user may join the protocol without requiring other users’ permission) and equi-privileged (all users have identical privileges). In this light, each of the above oracle solutions exhibits a strong form of centralization. Oraclize.it maintains a central server which handles all requests for off-chain information (giving it the privilege to deny requests, or to collude with website owners to produce false information). Town Crier also operates a centralized server with the notable difference that trusted hardware proofs are used to verify authenticity. Finally, though Augur includes a decentralized reporting/disputing mechanism, all markets must declare a Designated Reporter: a single Augur user with the privilege of reporting first on the market’s outcome.

**Different blockchain implementations**

We propose a novel paired-question protocol for decentralized oracles. When a user wishes to query the oracle, they submit two antithetic questions and post a bond. The oracle collects votes and checks whether the two questions converged to different answers; if so, the submitter regains their bond and voters are rewarded (penalized) for agreement (disagreement) with the majority answer. If the questions converged to the same answer, the submitter loses their bond and voters receive neither rewards nor penalties.

The interface for voters is simplified and truthful voters receive larger expected payoffs. We also introduce a general mathematical model of decentralized oracles and apply it to show that reporting one’s beliefs is a Nash equilibrium with better payoffs than in the lazy equilibrium.

**A Decentralized Oracle Model**

1) **Oracle Operator:** In order to provide decentralization it is assumed that the main operation of the protocol is handled on a suitably decentralized platform, such as through smart contracts on Ethereum [12] or Hyperledger [13]. This entity is hereafter referred to as the operator. The operator maintains a list of active Boolean propositions (i.e. statements which are either True or False). Any user may submit new propositions to the operator, and any user may vote on an active proposition. At certain times (as defined by the protocol) propositions are considered closed: the operator tallies votes, distributes rewards, and ceases to accept new votes on the losed

proposition.

2) **Voters**: In general, a decentralized oracle requires input from users. We say that a user who inputs data into an oracle is a voter, and could be any member of the public. Let V = {v1, v2, v3, ..., vn} be the set of all voters participating in the decentralized oracle protocol. For a proposition pj , each voter vi ∈ V has a private opinion POij ∈ {True, False}. This indicates what voter vi honestly believes about the validity of pj after reading its text. We assume the value of POij is fixed, but in the honest scenario it is unknown to voters other than vi. Voters who choose to collude and share their private opinions are considered adversarial.

Each voter vi also has a voting strategy σij , such that σij(POij) is the answer that vi actually reports to the oracle on proposition pj . For example, an honest voter has σi(POij) = POij , while a lazy voter has either σi(POij) = True for all j or σi(POij) = False for all j. Finally, we define two random variables: let Γj ∈ {True, False} denote the private opinion of a voter selected randomly from V on proposition pj , and let Aj ∈ {True, False} denote the answer reported by a voter selected randomly from V on proposition pj .

3) **Correctness of Oracle Outputs:** No corporeal entity (let alone a blockchain oracle) is capable of determining the objectively true answer to a question [14, §1]. Even in the elusive scenario where this were possible, users may choose to submit subjective questions (which do not really have an objective truth in the first place). Therefore, to rigorously define the notion of correctness in this work, we make the

following definitions:

Definition 1. **The Most Probable Private Opinion (MPPO)** is a randomly selected voter’s most likely private belief. On proposition pj , MPPOj

True P(Γj = True) > 0.5

False P(Γj = True) < 0.5

Undefined P(Γj = True) = 0.5

Definition 2. **We say that the oracle is correct on proposition** pj when its output is equal to MPPOj . Additionally, let cij denote voter vi’s perceived probability of reporting an answer to pj equal to MPPOj . Assuming that vi does not know POj for other voters, this is the same as the probability that vi answers MPPOj on a POj value selected randomly from V : cij .= P(σij(Γj) = MPPOj) (2)

We also let cj denote the probability that a voter selected randomly from V reports an answer equal to MPPOj : cj.= P(Aj = MPPOj) (3)

The key distinction between the definitions of cij and cj is that cij pertains to a specific voter vi with a fixed strategy σij , and thus may be different for voters with different strategies, whereas in cj the strategy also varies with the random variable

Aj .

As an example, if all voters in V adopt the honest voting strategy, then P(Aj = True) = P(Γj = True) and we have cj =P(Γj = True) MPPOj = True and P(Γj = False) MPPOj = False (4)

Because much of this formulation applies to any arbitrary proposition pj , we drop the subscript j from this point forward where no ambiguity arises from doing so.

**What Is the Nash Equilibrium?**

Nash equilibrium is a concept within game theory where the optimal outcome of a game is where there is no incentive to deviate from their initial strategy. More specifically, the Nash equilibrium is a concept of game theory where the optimal outcome of a game is one where no player has an incentive to deviate from his chosen strategy after considering an opponent's choice.

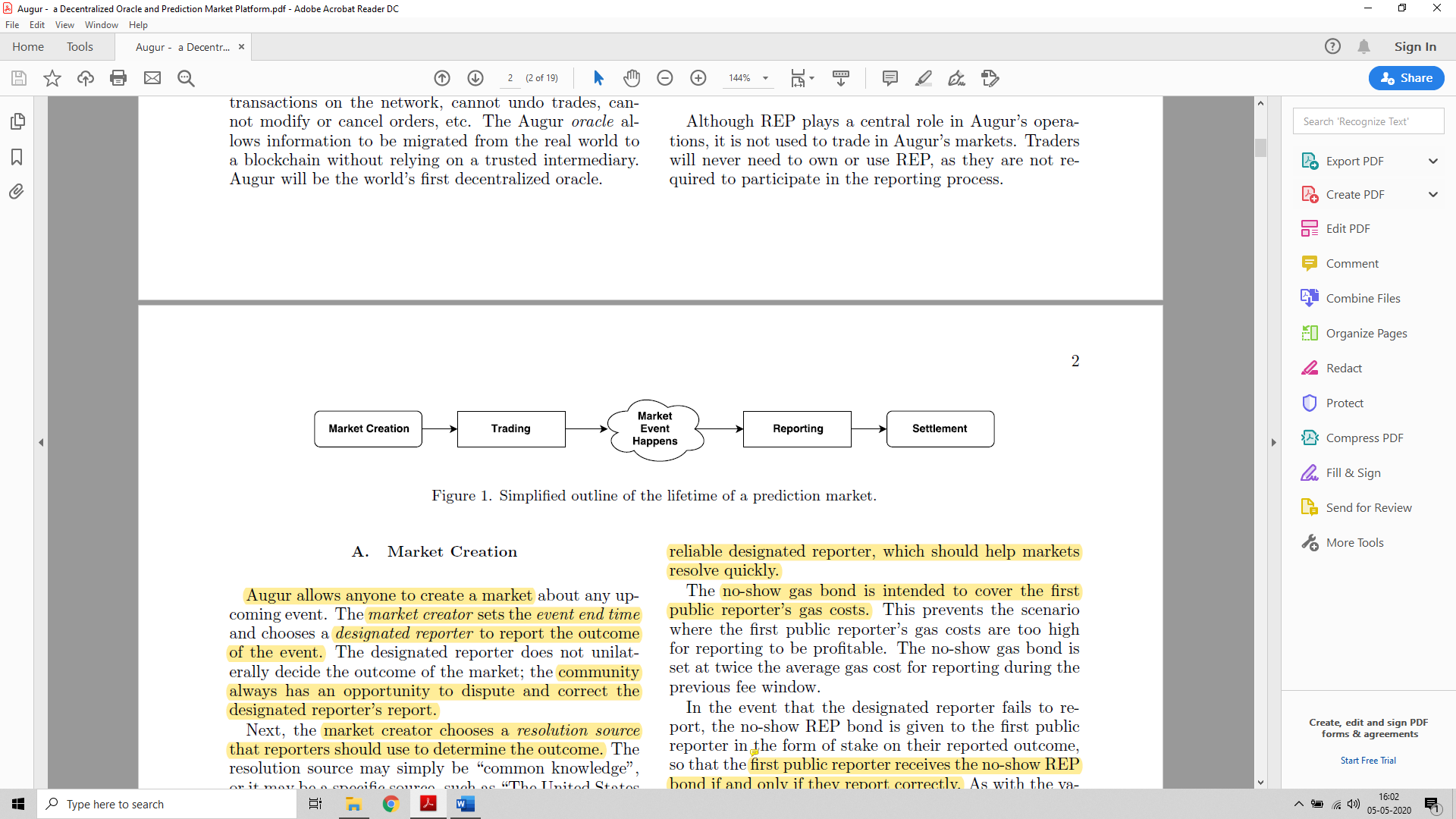
**Astrea:**

The outcome of voting is the stake-weighted sum of votes. Due to *the randomness involved when selecting a proposition, voting is resistant to manipulation by actors* seeking to force an incorrect result for a particular proposition. On the other hand, certifiers place large stakes on the truth or falsity of propositions of their choosing. Similar to voting, the outcome of certification is the stakeweighted sum of certifications.

In addition, we assume there are two certifier reward pools containing RT and RF monetary units, intended to reward certifiers for true and false outcomes, respectively. The use of two separate reward pools is intended to avoid a degenerate coordination strategy in which users always vote and certify with a constant true or false so as to maximize their profit without expending any effort. ??

**Note**: Certifiers’ higher stake doesn’t provide it a higher voting weightage, but only a higher return.

**Augur:**



A four-stage progression: *creation, trading, reporting, and settlement.*

Augur has a native token, Reputation (REP). REP is needed by market creators and by reporters when they report on the outcome of markets created on the Augur platform.

**Non-consensus:** Augur redistributes the REP staked on the non-consensus outcome by this reporter to the reporters that reported with the consensus.

**Augur’s market fees:** The more REP a reporter owns, and reports correctly with, the more fees they will earn for their work in keeping the platform secure.

**Market Creation:**

Augur allows anyone to create a market about any upcoming event. The market creator sets the event end time and chooses a designated reporter to report the outcome of the event. The community always has an opportunity to dispute and correct the designated reporter's report.

Two types of bonds submitted by - *validity bond and no-show bond*. The validity bond incentivizes market creators to create markets based on well-defined events with objective, unambiguous outcomes. The size of the validity bond is set dynamically, based on the pro-portion of invalid outcomes in recent markets.

The market creator forfeits the no-show bond and it is given to the first public reporter who reports on the market. This incentivizes the market creator to choose a reliable designated reporter, which should help markets resolve quickly. The no-show gas bond is intended to cover the first public reporter's gas costs. The first public reporter receives the no-show REP bond if and only if they report correctly.

**Trading:**

Augur's matching engine always sequesters the minimum amount of shares and/or cash needed to cover the value at risk. Orders are never executed at a worse price than the limit price set by the trader, but may be executed at a better price. Unfilled and partially-filled orders can be removed from the order book by the order's creator at any time. Fees are paid by traders only when complete sets of shares are sold.

For example, consider a market that has two possible outcomes, A and B. Alice is willing to pay 0.7 ETH for a share of A and Bob is willing to pay 0.3 ETH for a share of B. First, Augur matches these orders and collects a total of 1 ETH from Alice and Bob. Then Augur creates a complete set of shares, giving Alice the share of A and Bob the share of B.

**Reporting:**

Outcomes are determined by Augur's oracle, which consists of profit motivated reporters, who simply report the actual, real-world outcome of the event. Anyone who owns REP may participate in the reporting and disputing of outcomes. Reporters receive rewards in proportion to the amount of REP they staked during that fee window.

The participation token provides an incentive for REP holders to monitor the platform at least once per week.

**Designated Reporting Phase:**

Once the event end date has passed, the market enters the designated reporting phase. If the designated reporter fails to report within the allotted three days, the market creator forfeits the no-show bond and automatically enters the open reporting phase

If the designated reporter submits a report on time, then the no-show bond is returned to the market creator. The designated reporter is required to post the designated reporter stake on its reported outcome, which it will forfeit if the market finalizes to any outcome other than the one they reported.

**Open Reporting:**

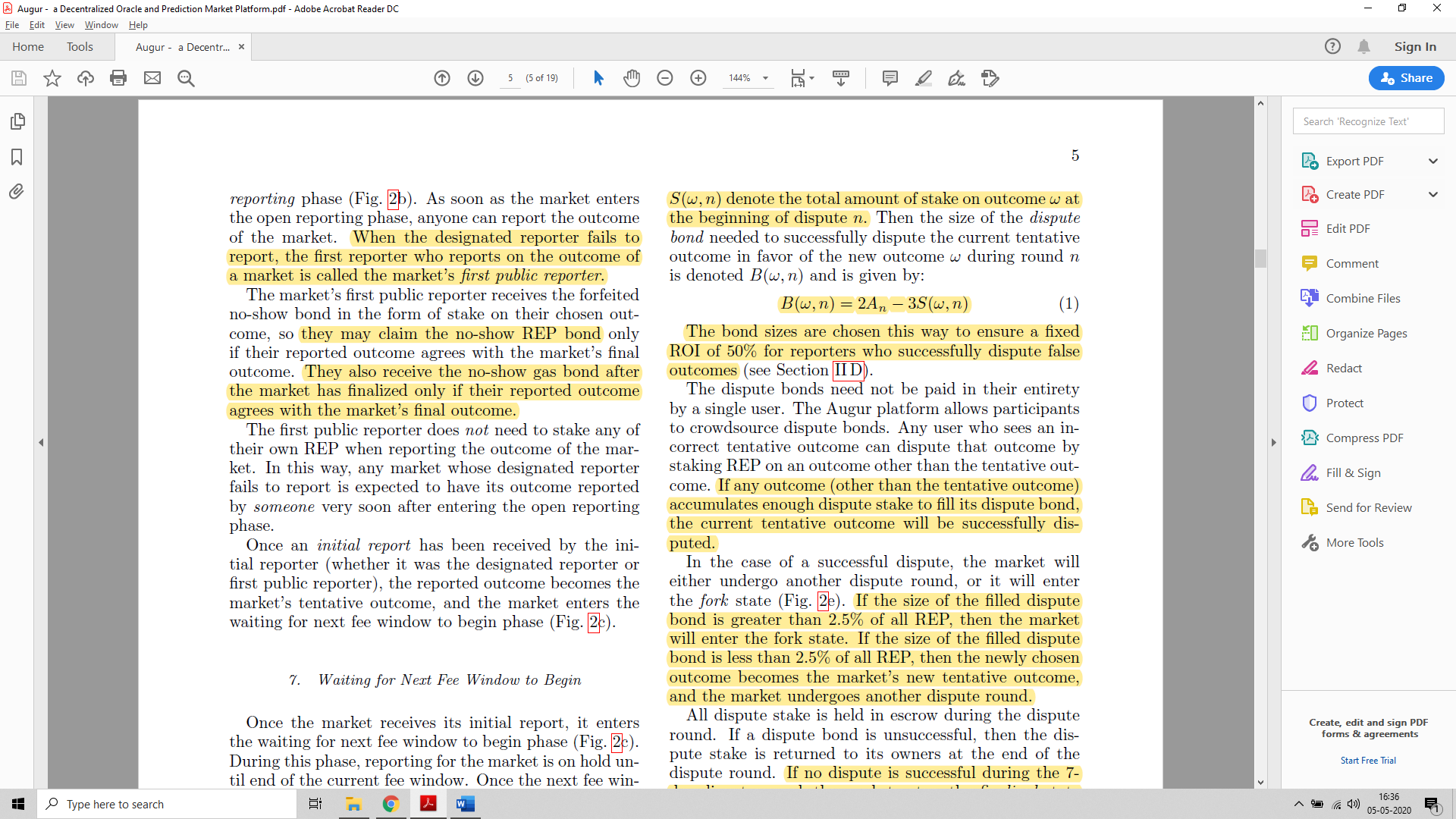
If the designated reporter fails to report within the allotted three days, the market creator forfeits the no- show bond, and the market immediately enters the open reporting phase.

**Dispute Round:**

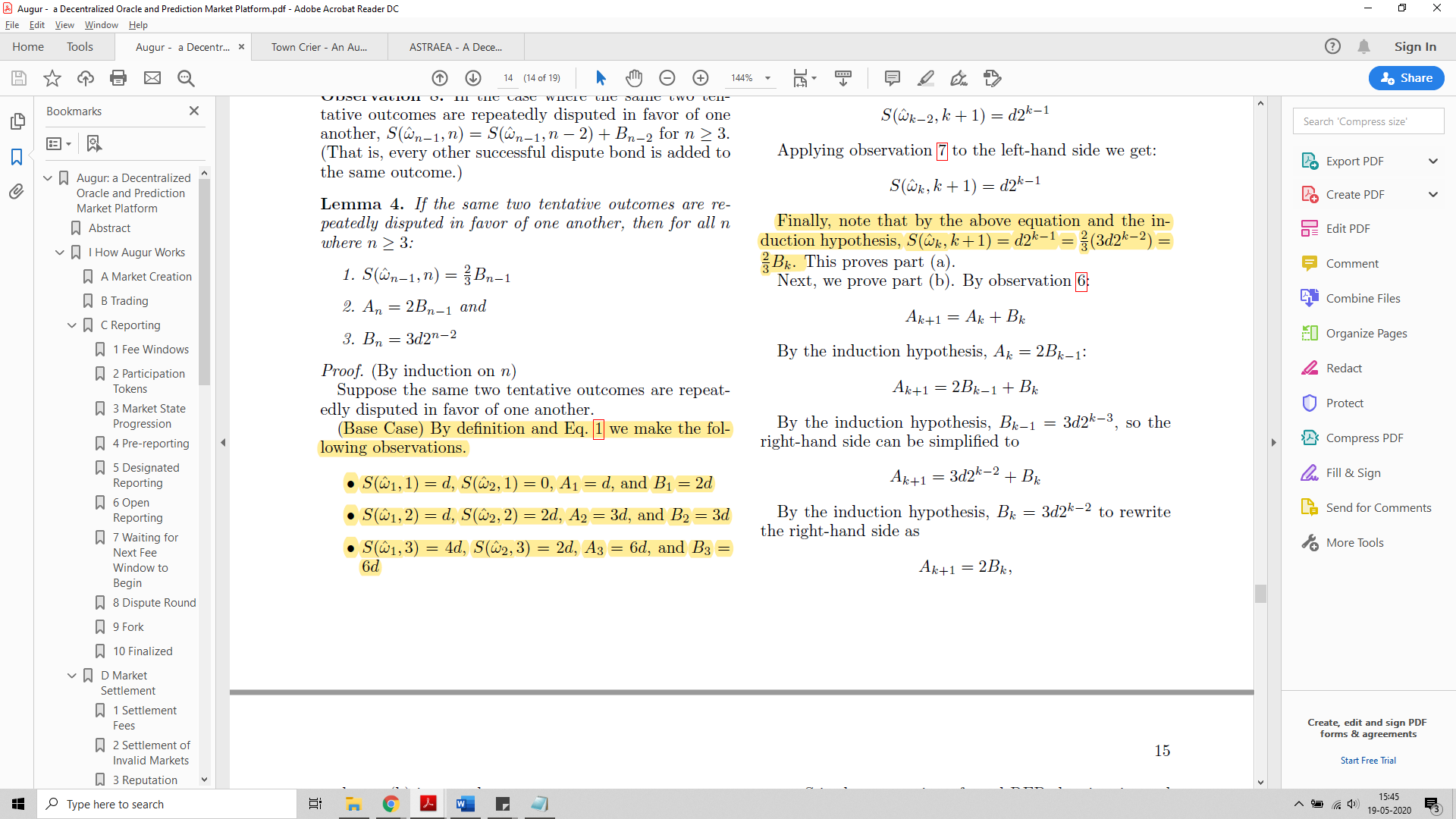
The dispute round is a 7-day period during which any REP holder has the opportunity to dispute the market's tentative outcome. A dispute consists of staking REP (referred to as dispute stake in this context) on an outcome other than the market's current tentative outcome.

**S(w, n)** denote the total amount of stake on outcome w at the beginning of dispute n. Let **An** denote the total stake over all of this market's outcomes at the beginning of dispute round **n**. **w** is an outcome other than the markets tentative outcome.

Then the size of the dispute bond needed to successfully dispute the current tentative outcome in favour of the new outcome w during round n is denoted B(w, n) and is given by:



The bond sizes are chosen this way to ensure a fixed ROI of 50% for reporters who successfully dispute false outcomes.



If the size of the filled dispute bond is greater than *2.5% of all REP*, then the market will enter the *fork state*. If the size of the filled dispute bond is *less than 2.5% of all REP*, then the newly chosen outcome becomes the market's new tentative outcome, and the *market undergoes another dispute round*.

All dispute stake is held in escrow during the dispute round. If a dispute bond is unsuccessful, then the dispute stake is returned to its owners at the end of the dispute round. If no dispute is successful during the 7-day dispute round, the market enters the finalized state, and its tentative outcome is accepted as its final outcome.

A ***market's final outcome*** is the tentative outcome that passes through a dispute round without being successfully disputed, or is determined via a fork.

**Fork:**

The fork state is a special state that lasts upto 60 days. When a fork is initiated, the parent universe becomes permanently locked. In a locked universe, no new markets may be created. Users may continue trading shares. Holders of REP tokens in the parent universe may migrate their tokens to a child universe of their choice.

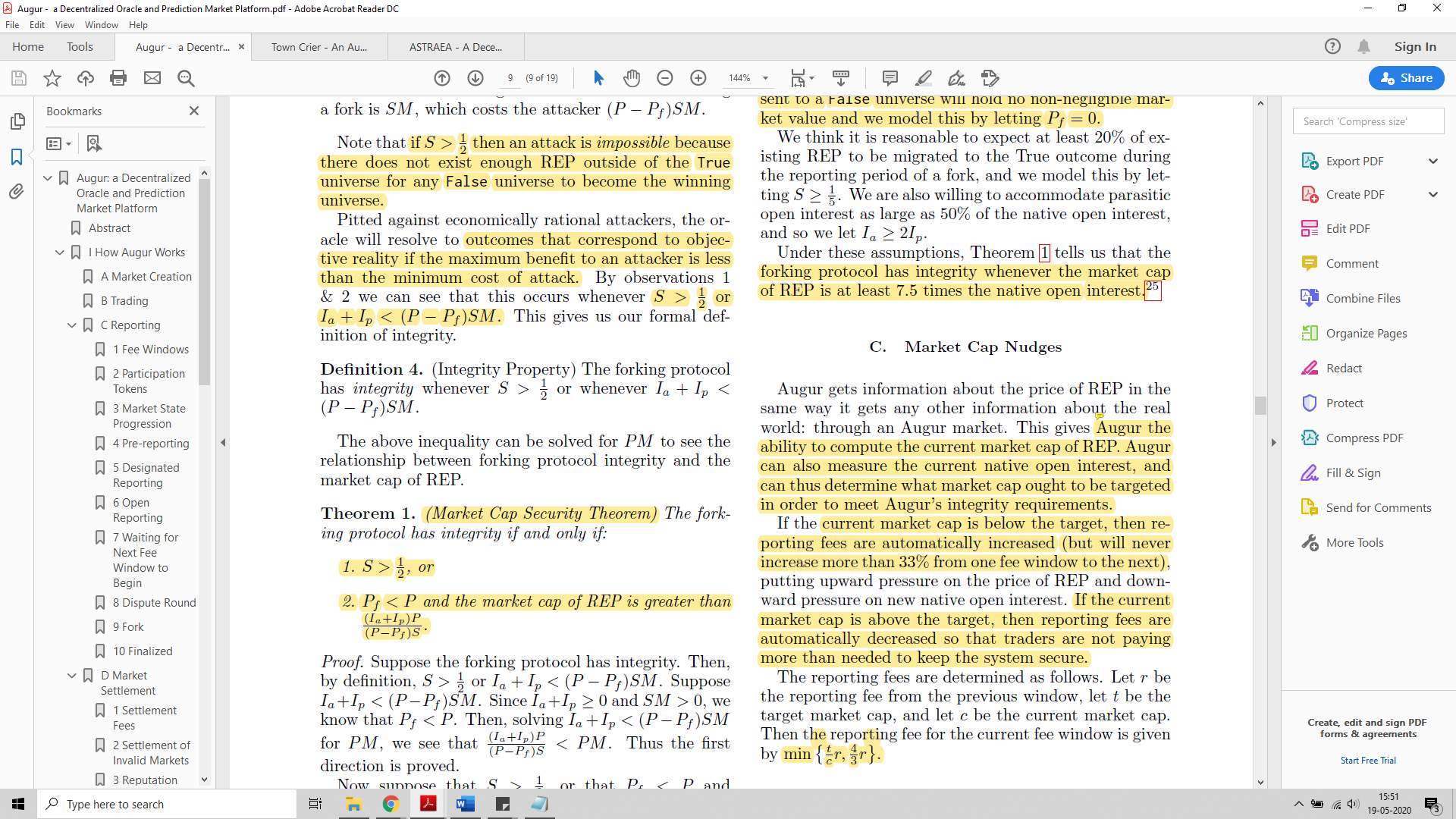
When a fork is initiated, all REP staked on all non-forking markets is unstaked so that it is free to be migrated to a child universe during the forking period. Whichever child universe receives the most migrated REP by the end of the forking period becomes the *winning universe.*

**Note**: Un-finalized markets in the parent universe may be migrated only to the winning universe.

There is no time limit to migrate tokens from the *parent universe* to a *child universe*. To encourage greater participation during the forking period, all token holders who migrate their REP within 60 days of the start of a fork will receive 5% additional REP in the child universe to which they migrated. This reward is paid for by minting new REP tokens.

**Note:** Reporters that have staked REP on one of the forking market's outcomes cannot change their position during a fork.

**Maintaining Integrity**



Here, S is the REP belonging to the True universe. There is not enough REP outside of the True universe for any False universe to become the winning universe.

Definition 1. We define, and denote by Ia, Augur's native open interest as the value of the sum of all funds escrowed in unfinalized Augur markets.

Definition 2. We define a parasitic market as any market that does not pay reporting fees to Augur, but does resolve in accordance with the resolution of a native Augur market.

Definition 3. We define, and denote by Ip, the parasitic open interest as the value of the sum of all funds escrowed in all parasitic markets that resolve in accordance to non-finalized, native Augur markets.

Assumptions:

it is reasonable to expect at least 20% of existing REP to be migrated to the True outcome during the reporting period of a fork, and we model this by letting S >=1/5 . We are also willing to accommodate parasitic open interest as large as 50% of the native open interest, and so we let Ia >= 2Ip.

Application:

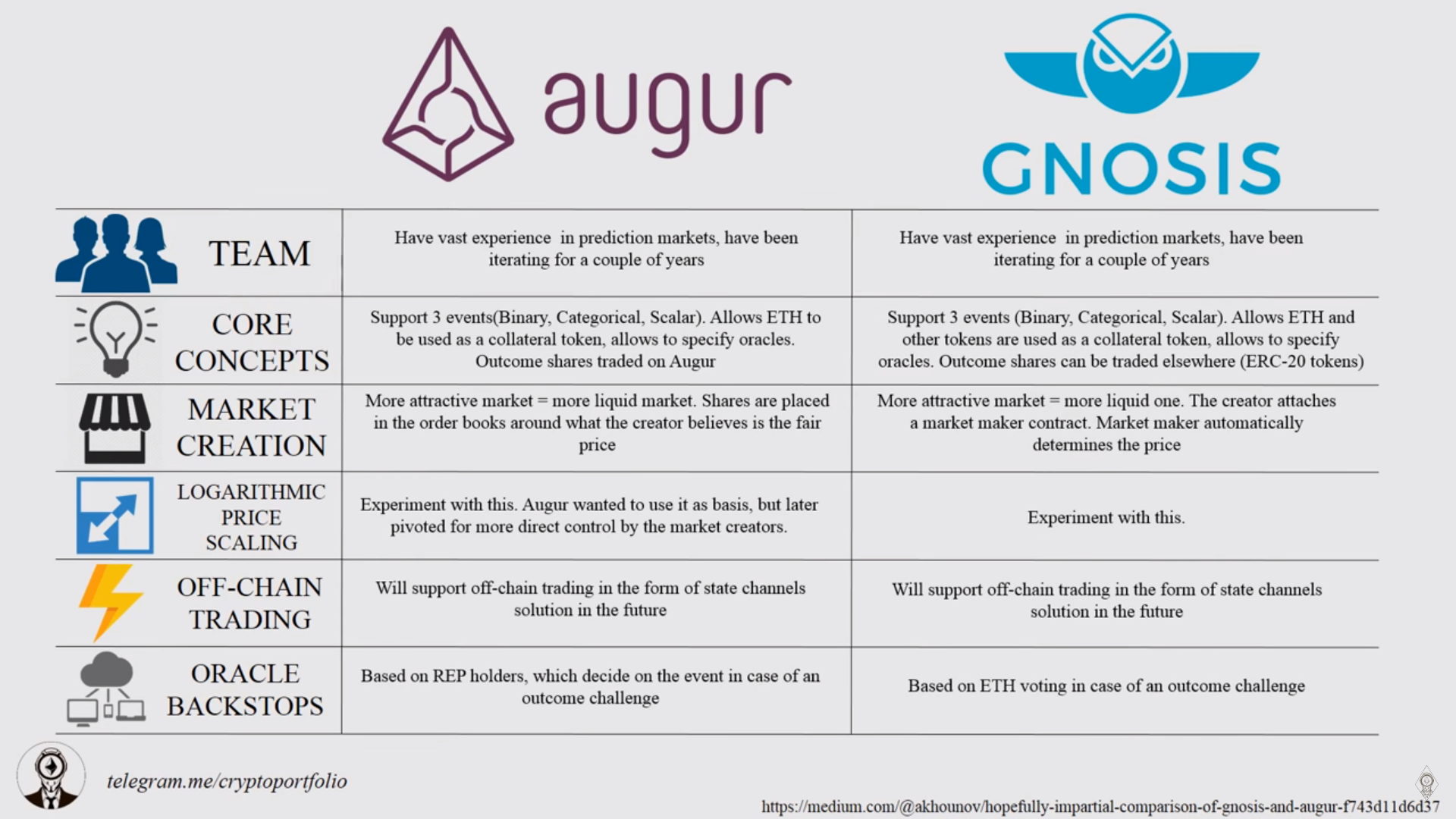
From the above assumptions we conclude – forking protocol has integrity whenever the market cap of REP is at least 7.5 times the native open interest. Augur has the ability to compute the current market cap of REP. Augur can also measure the current native open interest, and can thus determine what market cap ought to be targeted in order to meet Augur's integrity requirements.

If the current market cap is below the target, then reporting fees are automatically increased (but will never increase more than 33% from one fee window to the next). If the current market cap is above the target, then reporting fees are automatically decreased so that traders are not paying more than needed to keep the system secure.

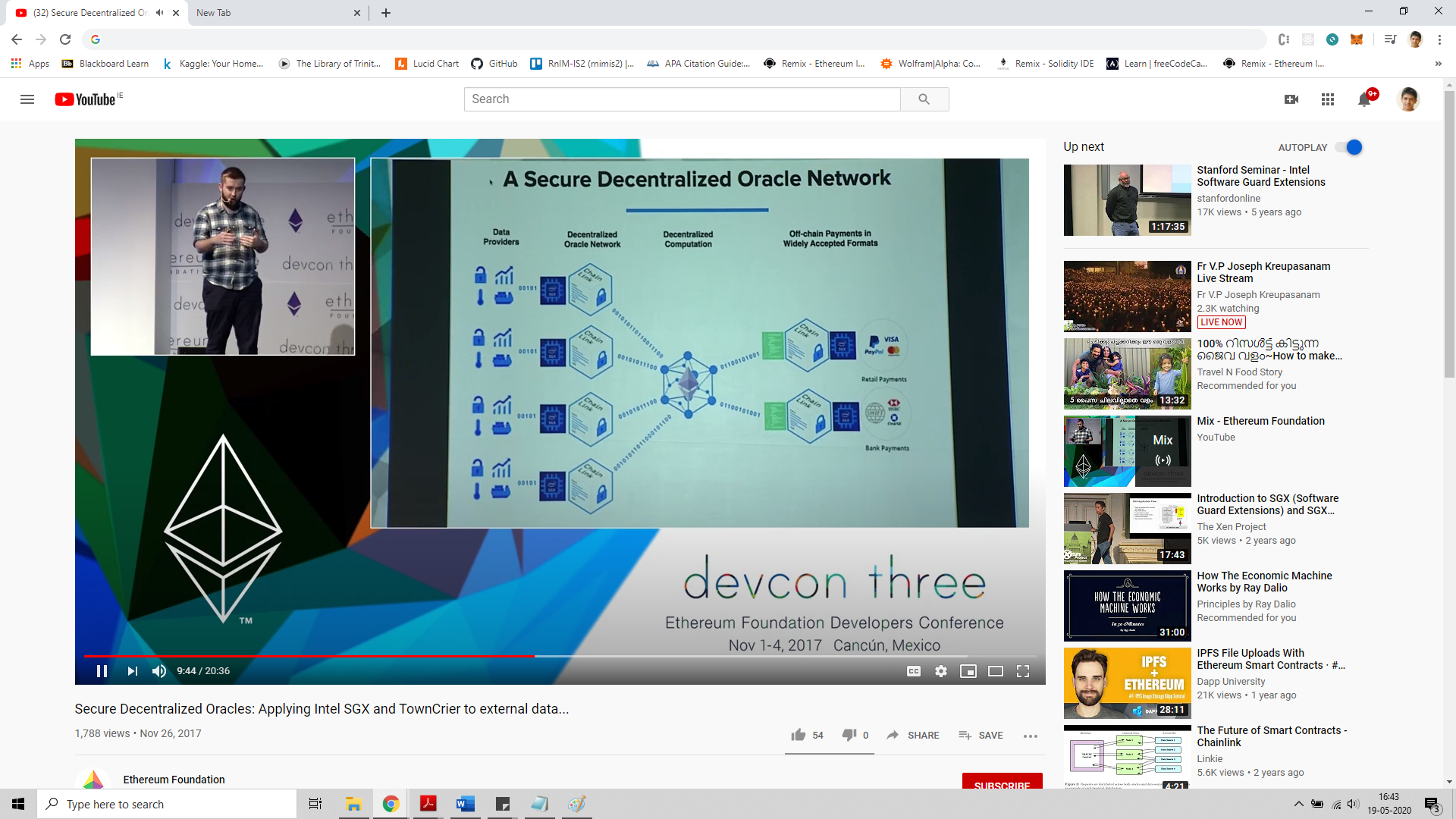
The reporting fees are determined as follows. Let r be the reporting fee from the previous window, let t be the target market cap, and let c be the current market cap. Then the reporting fee for the current fee window is given by *min{t/c\* r; 4/3\* r}*.

**Key points:**

Prediction markets allow participants in the trade to purchase and sell shares after the outcome of the event. Current market price of the share is an estimate of the probability of the event actually occurring. The price of each share adds up to one dollar. These markets rely on the principle known as **‘the wisdom of the crowd’** which states that if you ask people something their average answer is far more accurate than any expert.



Town Crier





Trusted Hardware:

Has a smaller trusted computing base. Privileged system code such as the OS and BIOS are no longer attack vectors for Intel SGX enclave. Provides for secured tamper proof oracles.

Town crier - Intel SGX that encrypts and makes data being sent immutable, has multiple nodes that suppy data to the contract(contract can select the number of nodes) , middleware that computes, off chain and on chain code/computations in SGX.

A combination of decentralization and trusted hardware.

SGX enclave has an EVM that computes and returns the byte code for the computation required by a contract in the blockchain. This facilitates ***off-chain computations***.

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>

Additional Data:

**Innovation –**

* middle ground, centralized oracles when data is usually from a single source
* Decentralized oracles when data is provided by multiple sources
* Machine learning to provide – models for threat detection, bigger incentives for right answers(especially trusted sources)
* **Nash Equilibrium**
* Randomness brought for voters as compared to certifiers in Astrea
* Trading reputation tokens, forking
* Intel SGX in Town Crier/ Chain link (immutable, signalling system, abstraction/hides important information from the node itself)

S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

Better understanding of hashing and merkel tree

"public keys encrypt, private keys decrypt" is correct... for data/message ENCRYPTION. For digital signatures, it is the reverse. With a digital signature, you are trying to prove that the document signed by you came from you. To do that, you need to use something that only YOU have: your private key.

A digital signature in its simplest description is a hash (SHA1, MD5, etc.) of the data (file, message, etc.) that is subsequently encrypted with the signer's private key. Since that is something only the signer has (or should have) that is where the trust comes from. EVERYONE has (or should have) access to the signer's public key.

So, to validate a digital signature, the recipient

1. Calculates a hash of the same data (file, message, etc.),
2. Decrypts the digital signature using the sender's PUBLIC key, and
3. Compares the 2 hash values.

If they match, the signature is considered valid. If they don't match, it either means that a different key was used to sign it, or that the data has been altered

Relationship between private key, digital signatures, public key

Blockchain and (Privacy + Regulation); ENIGMA (distributed hash tables, external blockchain that facilitates network, access and identity control); privacy by design concepts (data minimization, pseudominization, right to be forgotten and accountability), on-chain and off-chain data to implement PbD

Intel SGX - enclave(encrypted portion of the memory), encryption(data + code), decryption on the fly(happens within the CPU). Use cases - secure remote computation, secure web browsing, digital rights management, concealment of proprietary algorithms and encryption keys

**Digital Signature working:**

Trick here is to hide the private key within a function. In this case, (p-1)\*(q-1).

Consider p to be the private key and e to be the public key. 'p' is encapsulated within another function to make it hidden.

E.g., `d = (p-1)(q-1); d \* e = 1` (d is the inverse of e - public key)

Data sent = [encrypted(hash), message] = [m ^d, message]; where m is the message Suppose 'Data sent' = y To check for the integrity we find y ^e to get m. Since m ^(d\*e) = m ^1 = m.

**References**

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," 2014.

[3] A. Kiayias, A. Miller and D. Zindros, "Non-interactive proofs of proof-of-work," Cryptology ePrint Archive, Report 2017/963, 2017.

[4] J. Teutsch and C. Reitwiessner, "A scalable verification solution for blockchains," 2017.

[5] "TLSnotary - a mechanism for independently audited https sessions," 2014. [Online]. Available: <https://tlsnotary.org/TLSNotary.pdf>.

[6] . Zhang, E. Cecchetti, K. Croman, A. Juels and E. Shi, "Town Crier: An Authenticated Data Feed for Smart Contracts," Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016.

[7] J. Peterson, J. Krug, M. Zoltu, A. K. Williams and S. Alexander, "Augur: a Decentralized Oracle and Prediction Market Platform," 2018.

[8] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira and A. Kastania, "ASTRAEA: A Decentralized Blockchain Oracle," in IEEE Int'l Conference on Blockchain, Halifax, NS, Canada, 2018.

[9] L. Luu, J. Teutsch, K. Kulkarni and P. Saxena, "Demystifying incentives in the consensus computer," Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015.

Links:

<https://www.binance.vision/blockchain/blockchain-oracles-explained> - oracles in blockchain

<https://docs.chain.link/docs/create-a-chainlinked-project> - chainlink implementation (best)

<https://docs.augur.net/#overview> – Augur

<https://www.town-crier.org/dev.html> - Town Crier

<https://gnosis.io/pdf/gnosis-whitepaper.pdf> - Gnosis

town crier video - <https://www.youtube.com/watch?v=Zu6uZ1PnRU4>

<https://medium.com/coinmonks/using-apis-in-your-ethereum-smart-contract-with-oraclize-95656434292e>

<https://bitsofco.de/calling-smart-contract-functions-using-web3-js-call-vs-send/> - calling a smart contract using web3