ARCADA

Distributed IoT Security using an Ethereum-based Blockchain Infrastructure

John Wickström

Degree Thesis

Information Technology

2020

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	8065
Author:	John Wickström
Title:	Distributed IoT Security using an Ethereum-based Block-chain Infrastructure
Supervisor (Arcada):	Magnus Westerlund, DSc
Commissioned by:	

Abstract:

The recent proliferation of Internet of Things (IoT) has emphasized a lack of security and privacy in the industry. Many of the vulnerabilities stem from irresponsible practices committed by manufacturers and consumers alike and require a drastic paradigm shift to correct. The focus of this thesis is to understand how the Ethereum platform and block-chain technology can be utilized to bolster the security of the IoT sector. The proposed solution eliminates the need for trust by automating the workflow of devices through an interpreter that uses a distributed smart contract to determine its procedures. This form of indirect management allows a device to be hardened against internal threats by disallow-ing all incoming connections which significantly reduces the number of attack vectors that can be used to undermine a device's integrity. While this solution does intro-duce an unprecedented level of security to a technological sector that is in dire need of it, an in-evitable consequence is an increased cost of use and latency.

Keywords:	Ethereum, Blockchain, Smart Contract, IoT, Internet of Things
Number of pages:	44
Language:	English
Date of acceptance:	

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	8065
Författare:	John Wickström
Arbetets namn:	Distribuerad IoT-säkerhet med en Ethereum-baserad infrastruktur
Handledare (Arcada):	Magnus Westerlund, DSc
Uppdragsgivare:	
Sammandrag:	

Den explosivt växande populariteten av sakernas internet (IoT) har framhävt en stor säkerhetsbrist i teknologin. Både konsumenter och tillverkare har i långa tider använt sig av oansvarig praxis vilket tyder på att mer drastiska åtgärder måste vidtas för att korrigera det bakomliggande problemet. Syftet för detta examensarbete är att förstå hur Ethereum-plattformen och blockkedjeteknologi kunde tillämpas för att förstärka säkerheten för IoT-sektorn. Ett resultat kan uppnås med automation av en apparats arbetsflöde via en mellanvara som styrs av ett distribuerat smart kontrakt. En sådan form av indirekt styrning tillåter ägaren att härda en apparat mot interna hot vilket drastiskt minskar på mängden attackvektorer som kan användas för att angripa apparaten. Medan den förslagna lösningen definitivt kunde användas för att införa en hög nivå av säkerhet till IoT-branschen så är den dock också dyrare och långsammare att driva.

Nyckelord:	Ethereum, Blockkedja, Smarta Kontrakt, IoT, Sakernas Internet
Sidantal:	44
Språk:	Engelska
Datum för godkännande:	

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

1.1 Background

The conceptual idea of a fully electronic currency has existed since the 1990s, but all practical implementation attempts had very limited success and ended abruptly. However, in 2008 a person (or more likely a group of people) who remains unidentified but went by the pseudonym Satoshi Nakamoto published a whitepaper that claimed to have solved the issue of trust in a trustless system via cryptography (Nakamoto, 2008). What the paper detailed was the mechanics of the Bitcoin blockchain that started rapidly gaining traction. By the end of 2017, the price of one Bitcoin reached over 18000 USD after which the market suddenly crashed, and the price would drop by more than 500% within the next six months. Many perceived this as irrefutable evidence that cryptocurrencies are inherently flawed, even though the causality for the crash was consumerism rather than an algorithmic failure in the underlying technology.

Since Bitcoin's inception, a multitude of different blockchains have emerged that intend to solve prominent issues like computation, storage, and direct messaging in a distributed environment by utilizing Nakamoto's consensus model. A prime example of this is the Ethereum blockchain that allows snippets of code (later named smart contracts) to be run on a decentralized virtual machine (Buterin, 2014). Applications that exclusively use decentralized components are referred to as decentralized applications (dApps) and symbolize how the third incarnation of the web could function.

The recent rise in popularity of Internet of Things (IoT) has been explosive and shows no signs of stagnation. Unfortunately, online services like Shodan (Shodan.io, 2013) that find vulnerable devices by scouring the Internet for open network ports prove that a shocking amount of IoT devices are insecure and can easily be tampered with. While some government-mandated standardization research has been started to mend this issue, these cogs turn very slowly and the burden to resolve security and privacy flaws in the short-term falls on the shoulders of the consumer.

1.2 Objective

The objective of this thesis is to explore the Ethereum platform and its protocols to understand how they can be leveraged to increase the security and privacy of the IoT sector. After a thorough exploration of key concepts, the discussed principles will be showcased in a practical setting by detailing the architecture of the following components:

- A distributed smart contract backend system designed for blockchain oracles.
- An interpreter that allows an IoT device to perform actions autonomously and to be managed indirectly through a smart contract.
- A service discovery system for IoT devices that utilizes the Ethereum platform's Whisper protocol.
- A user-friendly client to customize a smart contract oracle's objective.
- A web-application that streamlines the usage of the smart contract backend.

Each of these components are part of a larger system that will be discussed on a relatively high level in section 6. The detailed source-code can be viewed from the respective components GitHub repository:

Smart contracts:	https://github.com/wickstjo/oracle-manager
IoT interpreter:	https://github.com/wickstjo/iot-manager
Web-application:	https://github.com/wickstjo/distributed-task-manager

1.3 Delimitation

This is first and foremost an exploration of the Ethereum platform and its protocols, so other platforms and blockchains, however similar, will not be explored. Because Ethereum's 2.0 (Ethereum Foundation, 2020) version is in its very early stages, only features exclusive to the 1.x versions will be used. Smart contracts will be written in the Solidity programming language (docs.soliditylang.org, 2020) and utilized in a JavaScript and Python environment. The IoT device that is used for the practical implementation is a Raspberry Pi 2B (Raspberry Pi Foundation, 2019) microcontroller.

2 GENERAL CONCEPTS

2.1 Transcoding

A transcoder is an algorithm that can convert one type of data into another without losing its structural integrity. This technique is often used in complex applications that consist of multiple different codebases that pass incompatible data between modules. The transcoding process is often referred to as encoding, while the reversal of the effect is referred to as decoding.

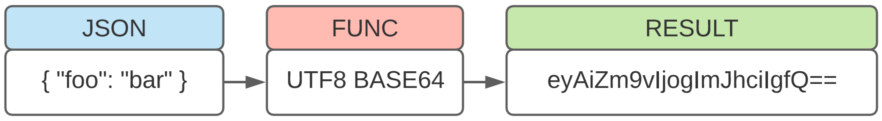

Figure 1: Transcoding a JSON object with the base64 algorithm.

Generally transcoded data loses a small segment of the original content due to compression, but there are plenty of algorithms that allow for lossless transcoding at the cost of efficiency. In the implementation section of this thesis, we will be utilizing a base64 transcoder to pass lossless JSON data between submodules and the blockchain as presented in Figure 1.

2.2 Hashing

Hashing refers to an action where an arbitrary amount of data is put through a specific algorithm to convert it to an indecipherable string of fixed length. Unless the hash algorithm is cracked and reverse-engineered, there is no way (other than guessing) to reconstruct the data that was used to generate a specific hash.

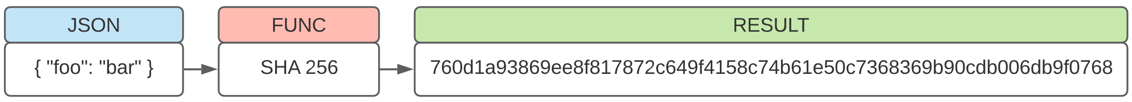
JSON
{ "foo": "bar" }
FUNC
SHA 256
RESULT
760d1a93869ee8f817872c649f4158c74b61e50c7368369b90cdb006db9f0768

Figure 2: Hashing a JSON object with the SHA256 algorithm.

This is an extremely powerful tool in programming because you can snapshot the content of anything from simple data structures to huge programs and condense them down to a virtual fingerprint. These fingerprints that are also known as checksums are only a few bytes long but can be fully relied upon to verify that the source material has not been tampered with in any way. In the implementation section of this thesis, both indexing and verification procedures use hashing as presented in Figure 2.

2.3 Encryption

Encryption, much like hashing, starts from data being sent into a specific algorithm to be turned into an undecipherable string. However, encrypted data has the unique capability of reverting itself to its original state if the correct key (or keys) are first provided. The two main types of encryption are symmetric and asymmetric.

2.3.1 Symmetric Encryption

For symmetric encryption, the same key is used to encrypt and decrypt a file. This is the preferred encryption type for files that are distributed to multiple users simultaneously but do not contain information that is sensitive enough to warrant the added complexity of asymmetric encryption.

2.3.2 Asymmetric Encryption

For asymmetric encryption, two different sets of public and private keypairs are used in conjunction to encrypt and decrypt a file. In the scenario that is depicted in Figure 3, user A uses their private key and user B’s public key to encrypt a file. After receiving the encrypted file, user B would use their private key and user A’s public key to decrypt and read the file. Note that under no circumstances should two users ever exchange private keys.

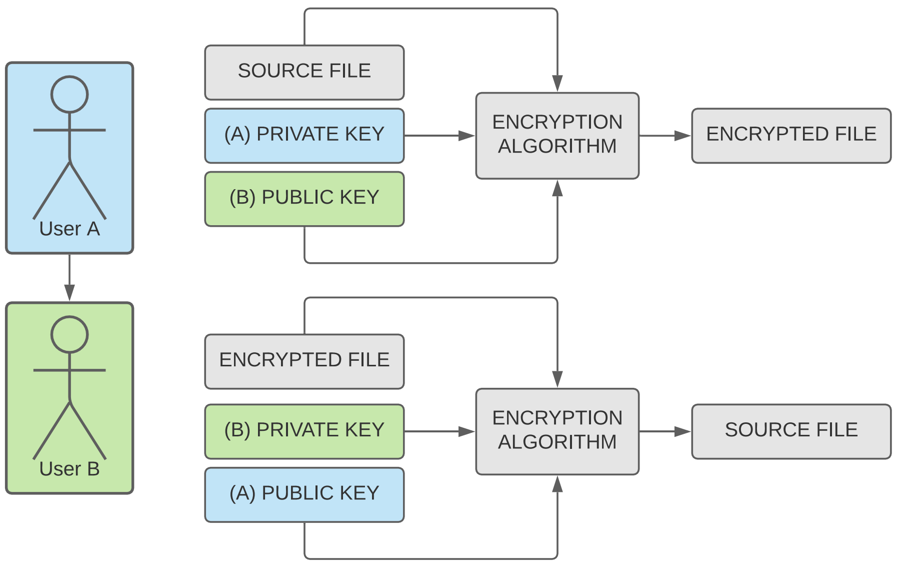

Figure 3: Users A and B asymmetrically encrypt and decrypt a file.

Depending on the asymmetric encryption algorithm, keypairs can either be generated for each subsequent encryption on the spot or fetched from a publicly available database like Pretty Good Privacy (Garfinkel, 1995) for continuous use of the same keypair. The first method provides a higher level of security because a compromised keypair can only be used to decrypt a single file, but requires users to manage a potentially infinite number of keypairs. The second method delegates keypair management to a third party, but a compromised keypair would mean that all the files associated with it can be decrypted.

2.4 Network Governance

There are typically considered to be three categories of network governance: centralized, decentralized, and distributed (see Figure 4). None of them are fundamentally superior to the others because they promote and value entirely different things and therefore suit different scenarios. A system or network is not bound to only one type of governance and often thrives by utilizing a combination of two or even three at once. (Nguyen et al. 2016)

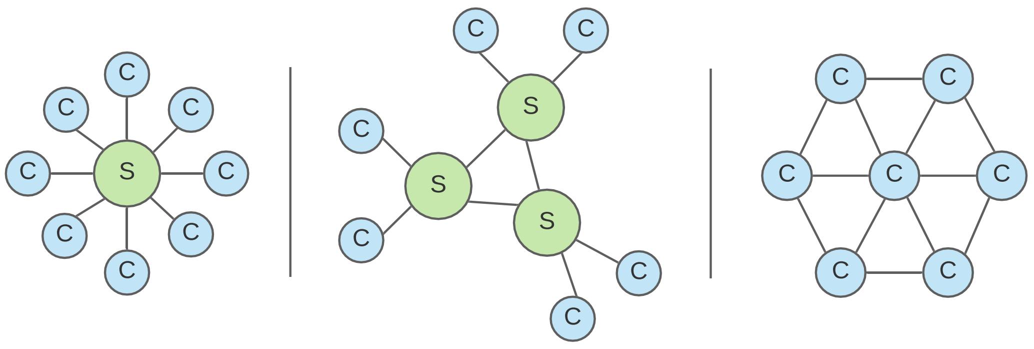

Figure 4: Centralized, decentralized, and distributed networks visualized.

The key component of a centralized network is the central authority (a central point of failure) and a disproportionally large number of nodes connected to it. A prime example of this would be a typical internet service provider as the central authority with its clientele as the connected nodes.

In a distributed network, every node is interchangeable because they all either contain the same data or can collectively reconstruct the same data by borrowing missing blocks from neighboring nodes. It can be argued that this category of governance is the only one that can truly be called democratic because there is no central authority that controls any part of its content or mandates how it should be operated.

A decentralized network is a subset of a distributed network where no single node has absolute authority over the others, but there is a significant disparity in node connectivity and size. In other words, it is a hybrid governance model that is used by large companies all over the world. It would be unimaginable, likely even impossible, for a company like Amazon with clients and partners all over the world to be run in a fully centralized fashion. Instead, there are multiple servers that clients can connect to that all serve more or less the same content. If one of the servers were to fall, the clients would be forwarded to another server. Regardless, Amazon is still the central authority of the network and can control its content, hence the hybrid categorization.

2.5 Blockchain Technology

A blockchain is a continuously growing database of transaction records that are stored in chronologically ordered blocks which are interlinked by cryptography. Every subsequent block post genesis contains at the very least:

1. Its own subset of transactions.
2. A unique hash (block hash) that is derived from its transactions.
3. A block hash reference to its predecessor to establish an ordered hierarchy.

Because of this, a blockchain is considered to be immutable because retroactive modification of any block in the ordered hierarchy also requires the modification of every subsequent block after it. The number of blocks grows linearly over time based on a static hash rate, which means that a blockchain becomes more robust and difficult to break with maturity. (Zheng et al. 2017)

A blockchain is managed by a peer-to-peer network where every node has access to the entire transaction history and can therefore trace every unit of currency back to its inception. This type of network is often referred to as a distributed ledger. One of the primary problems for such a network is to establish computational consensus among nodes, a rule of law if you will. Nakamoto (2008) proposed that this problem could be solved by a so-called consensus algorithm.

2.5.1 Consensus Algorithms

A consensus algorithm allows a blockchain to be managed by a distributed network of inherently untrustworthy peers by dictating their conduct. There are several different algorithms that build on fundamentally different principles and assumptions about human behavior, but since this thesis focuses on the Ethereum blockchain, only the Proof of Work (PoW) and Proof of Stake (PoS) algorithms which are relevant to it will be discussed. A blockchain cannot adhere to multiple consensus algorithms simultaneously but can transition from using one to another.

2.5.2 Mining

Transactions are not arbitrarily appended to a block and must first be validated by a special type of network peer that is called a miner. The computation that is necessary for this validation process is referred to as mining and its methodology is dictated by the blockchains consensus algorithm. A majority of network nodes must approve the validity of a mined transaction. If the network nodes approve, the miner is rewarded with newly created cryptocurrency and the transaction will be appended to the next block. Mining is done completely voluntarily, so every node does not have to be a miner, but every miner has to be a node.

2.5.3 Proof of Work

When a blockchain is managed by a Proof of Work (PoW) consensus algorithm, the miners compete against each other to solve a computational puzzle to create new blocks. To solve the puzzle, a miner must first guess a pseudo-random number that is referred to as a “nonce”. The generated nonce and the blocks transactions are then simultaneously passed through a hash function to produce a “block hash”. This process is repeated until the block hash meets the blockchains PoW complexity requirement. (Gervais et al. 2016)

When a miner finds a nonce that can be used to produce a sufficiently complex block hash, they publish it to the network where the other nodes check its validity by hashing the block’s transactions with the provided nonce themselves. If the validating nodes accept the result, the miner is rewarded and a new block that contains these parameters will be appended to the chain. Note that to find a nonce that produces a sufficient block hash is the labor-intensive part of the problem, while the verification of it is trivial in comparison.

According to Cambridge Centre for Alternative Finance (cbeci.org, 2020), the cumulative energy consumption of every Bitcoin (PoW) miner is equivalent to a small country and only a fraction of the computation is truly necessary. PoW has been computationally proven to be reliable which is why most of the highest valued blockchains use it, but the environmental consequences might lead to its eventual downfall.

2.5.4 Proof of Stake

With a Proof of Stake (PoS) consensus algorithm, miners (called validators) are no longer required to solve a computational puzzle and instead only validate the transactions of a block and stake currency as a guarantee of their validity. Validators are chosen through a pseudorandom election process based on different factors and characteristics. The reward for validating a block is equal to the fees of its containing transactions. (Li et al. 2017)

For a node to become a validator, it must first stake a minimum amount of currency that the network seizes. When the PoS algorithm chooses a validator, it can take into consideration the quantity, weight, and age of a node's staked assets. An asset's weight is determined by how long the owner is willing to forfeit it for, while its age refers to how long it has been since the owner was last selected for validation. To prevent wealthy nodes from perpetually being selected, some randomized factors are also taken into consideration by the algorithm. A node's accumulated rewards are added to the initial stake and will remain seized by the network until the node wants to stop validating transactions and the forfeit duration has expired.

Validated blocks still have to be approved by the majority of the network's nodes. If a validating node approves transactions that the rest of the network considers to be invalid, then the node is immediately punished by destroying a part of its staked assets and disqualifying it from being chosen to become a validator again. For as long as a node's staked assets are larger than the reward for a block, there is an incentive for the node to conduct itself truthfully.

2.5.5 Forking

A blockchain is an extremely complex ecosystem requiring updates and modifications from time to time. This can be due to an exploit being found that needs to be subsequently patched or because the development team has scheduled the implementation of a specific feature far into the future. However, because of the distributed nature of blockchains, not even the creators have the ability to force a change onto the existing node network. To deploy a change, an entirely new blockchain must be formed in a process called forking. A forked blockchain contains a carbon copy of the original's transaction history, so any currency or content that existed on the former will also exist on the latter. Figure 5 shows a blockchain fork at block number six.

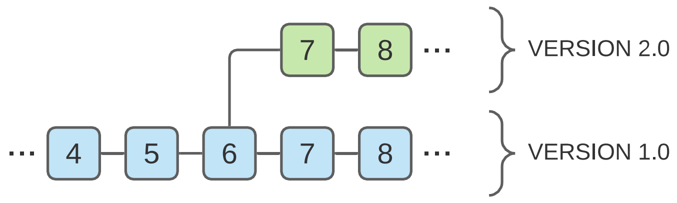

Figure 5: A blockchain is forked to version 2.0 at block number 6.

Forks can be categorized as soft or hard depending on their cross-compatibility with the original. A soft fork occurs when the included update is small, and users with different forks can still interact with each other. A hard fork occurs when a foundational modification is included, and the two networks cannot be used interchangeably. Note that there is no forceful transitioning of nodes and anyone who wants to keep using an older fork is free to do so. The only thing that gives a blockchain any kind of market value is its users' trust in the integrity of the system.

3 ETHEREUM

After a successful crowdfunding campaign, the Ethereum platform went live in July of 2015. Initially, it only had the blockchain protocol (Buterin, 2014), but over time the Whisper and Swarm protocols have been added that intend to solve completely different problems in the distributed ecosystem and pave the way for the third incarnation of the web. The Whisper protocol (Ethereum Wiki, 2020) functions very similarly to the message queuing telemetry transport protocol (Shinde et al. 2016) and is intended for short, encrypted peer-to-peer messaging. The Swarm protocol (Trón et al. 2020) deals entirely with distributed storage and functions as a content delivery network (CDN) that is reminiscent of the BitTorrent protocol. (Pouwelse et al. 2005)

Since the Swarm protocol is still in very early development on the Ropsten test network, this thesis will only discuss and utilize Ethereum's blockchain and Whisper protocols.

3.1 Blockchain Protocol

3.1.1 Ether

The Ethereum blockchains cryptocurrency is called Ether and has seven different denominations of which the smallest is called a Wei and represents 1e-18 Ether. This currency can be traded for traditional commerce and services, but also as payment for a special mining reward called gas that can be increased to prioritize a transaction's incorporation to the next block.

3.1.2 Accounts

Reading data from the blockchain can be done anonymously, but to write new data or modify existing data through either conventional currency trading or a smart contract interaction requires an Ethereum account. This unique account consists of a public/private keypair that is used to authorize transactions. A user's public key will be attached to every transaction they make and is safe to be shared with other users, but the private key should under no circumstances be shared with anyone else.

Humans are notoriously bad at remembering long randomized alphanumeric strings like encryption keys, so to alleviate this problem, accounts are commonly created by using a mnemonic seed. A mnemonic seed, as depicted in Figure 6, is a set of randomly chosen words from the BIP 32 English wordlist (Wuille, 2012) that can be used to reconstruct an Ethereum accounts keypair. Users can also delegate keypair management to a separate wallet application like Metamask (Metamask.io, 2019) to authorize transactions effortlessly and securely in a web browser or on a smartphone.

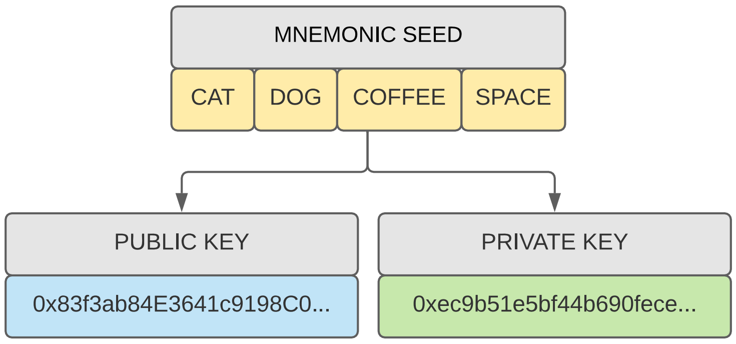

Figure 6: An Ethereum accounts mnemonic seed.

3.1.3 Ethereum Virtual Machine

Most blockchains only handle and archive transactions which is why they are often referred to as distributed ledgers. However, the Ethereum blockchain also has the unique capability to execute machine code in the form of smart contracts on a decentralized virtual machine called the Ethereum Virtual Machine (EVM). Instead of containing transactions like other distributed ledgers, the Ethereum blockchain contains the state history of the EVM. A more appropriate description for the Ethereum blockchain would be a distributed state machine that contains transactional data. (Buterin, 2014)

3.1.4 Smart Contracts

Smart contracts are snippets of code that are executed on the Ethereum virtual machine (EVM) and are an integral part of the Ethereum blockchain. These contracts can be written in multiple programming languages that resemble either JavaScript, Python, or Lisp, since the result will ultimately be compiled down to EVM bytecode before deployment.

The immutable property of the Ethereum blockchain is also extended to its smart contracts. What this means programmatically is that new variables and functions cannot be added after the contract has been deployed. However, existing variables can be modified and dynamic data structures like lists and hash tables allow for pushing and removal of values if the respective setter functions have been declared. Versioning works similarly to Git (Git-scm.com, 2019) in that every previous state of a smart contract is permanently cataloged but only the most recent state is presented to users by default. Immutability puts an enormous amount of pressure on developers and their unit testing capabilities because bugs and exploits that are found post-deployment cannot be fixed. Smart contract auditing has become a big business with an increasing amount of money and effort being spent on attempts to automate the entire process. Tools like Mythx (mythx.io, 2020) can crawl through contract code and compare it to known exploits and vulnerabilities listed in a publicly available database called the SWC Registry (swcregistry.io, 2020) that the smart contract community maintains together.

Smart contracts and the EVM do not have access to any information that does not exist on the blockchain. This means that all conditional statements within a function need to be measurable and deterministic in nature so network peers can come to a consensus. However, external data can be injected and made available on the blockchain through an oracle design pattern that will be discussed further in section 4.1.

3.1.5 Global Namespace

Smart contracts have a short list of special variables that are globally available and cannot be falsified or overwritten. These variables return information about the most recently mined block or the transaction that triggered the computation.

The block variables are often used to measure the passage of time through either a Unix timestamp or a block number. Some casino applications that use a smart contract backend have mistakenly used block variables as a source of pseudorandomness in gambling. Because the Ethereum blockchain has a static hash rate that dictates how often new blocks are created, new block parameters can be predicted with a relatively high probability. Even imperfect predictions can be enough to frequently derive the result of a pseudorandom number generator that uses block variables in an equation.

The transaction variables are used in almost every smart contract. They provide an unfalsifiable method of checking fundamentally important properties like the public key of the Ethereum account that executed a function or how much Ether was provided with the transaction. The execution of entire functions can be limited to authorized users or other smart contracts through this methodology.

3.1.6 Transactions and Gas Consumption

Any smart contract function that adds, reduces, or modifies data inside the contract itself or any other contract constitutes a transaction that must be mined and validated by network peers. The Ethereum account that authorizes the execution of a function must also provide enough Ether to cover its computational cost before the EVM propagates the transaction to miners. All transactions generate a receipt hash that can be input into a service like Etherscan (Etherscan.io, 2019) to track its validation progress.

The reward for miners on Ethereum is called gas. It is a transaction property that can be increased beyond its minimum value to make a transaction more profitable for miners, so they prioritize its incorporation to the next block over other transactions. The price of a single unit of gas fluctuates frequently because it operates via supply and demand, so during primetime when traffic is high, the price of gas will be more expensive. The current price of gas can be monitored from the blockchain directly or via web services like Ethgasstation (Ethgasstation.info, 2016) that track it.

Every transaction has a minimum gas value that correlates to how much computation is required to mine it. A common mantra among smart contract developers is "complexity is penalized" because solving a problem through bloated and ineffective code will inherently cost more gas to execute compared to a short and efficient solution. Everything from imports to data structure types need to be considered when optimizing for gas consumption. Functions that are executed frequently should be designed to be as light as possible while functions that are rarely executed (like constructors) should be responsible for transporting heavier payloads.

3.1.7 Networks

Most blockchains only have one network, but because of how punishing and irreversible committed mistakes are for smart contracts, Ethereum has five different officially supported networks. The first one is called the "mainnet" and is the one true version that the Ethereum blockchain operates on. The four remaining ones are testing environments with different settings like consensus algorithms where developers can conduct experiments. The most like-to-like “testnet” to mainnet is called Ropsten and operates with a proof of work consensus algorithm.

Testnets work similarly to the mainnet and require cryptocurrency to use. While this currency can be mined, most developers use something referred to as “network faucets” instead. These are simple web applications made by altruistic users that donate currency to other users upon request. Testnet currency has no real trade value since the entire network only exists as a testing environment.

3.1.8 Synchronization and Gateways

Since the blockchain is governed by a distributed network of nodes, you need to become a node in that very same network to interact with it. This can be done by a process called synchronization which essentially means downloading the blockchain’s entire transaction history. Note that because the number of blocks in a blockchain grows linearly over time, so will the required storage space for a full synchronization.

According to Etherscan (Etherscan.io, 2019), the current storage requirement for a full node synchronization is 570 gigabytes. This storage requirement is slowly getting out of hand even for desktop computers with large hard drives, which means that handheld devices are effectively excluded from being able to synchronize and use the blockchain. To work around this problem, devices can delegate synchronization to another machine and interact with the blockchain through it. These types of services are called gateways (see Figure 7) and are simple to set up yourself but can also be purchased as a web service from companies like Cloudflare (Cloudflare, 2020) or Infura (Infura, 2020).

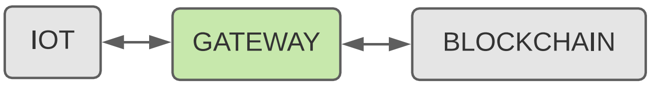

Figure 7: An IoT device interacting with the blockchain via a gateway.

As convenient as gateways are, it is important to understand that they can introduce an uncomfortable level of centralization to a system that strives to be fully distributed. Foreign gateways should be used responsibly, meaning that relatively small and insignificant transactions can be pushed via them, but large and important ones should not. A reasonable compromise is to primarily push transactions via your own gateways and only use foreign ones as a backup if something catastrophic were to happen.

3.2 Whisper Protocol

The Ethereum platform has many different protocols that independently deal with a specific subset of the distributed technology stack. While the blockchain protocol deals with distributed computation, the Whisper protocol is intended for short, encrypted peer-to-peer messaging and is reminiscent of the message queuing telemetry transport protocol (Shinde et al. 2016). Whisper is entirely identity-based and is not influenced by any hardware attributes or characteristics. The protocol was built from the ground up with security and anonymity in mind of which both can be increased at the cost of increased latency. (Ethereum Wiki, 2020).

Like Internet Relay Chat (Oikarinen, 1993), Whisper users can send messages to each other directly or to a shared channel where the message can be read by all channel members. To connect to a Whisper channel, which is called a topic, a user must first provide a 4-byte long name and the correct symmetric encryption key before the contained messages can be deciphered. Direct messaging between users utilizes an asymmetric encryption scheme where the two users encrypt and decrypt messages with public and private keys. Encryption keys for identities and topics are generated directly through the protocol's application programming interface (API).

3.2.1 Darkness

Most communication protocols are endpoint-based because it allows for messages to be sent with minimal latency. An unavoidable consequence of this methodology is that the protocol's network traffic is trackable and links between nodes begin to form a pattern that can be used to extrapolate further information about the nodes. Whispers solution for this problem is to propagate messages probabilistically to every node in the network. This gives users plausible deniability because no message can be traced back to a single user or node. The described feature denotes darkness and comes at a significant cost. See Figure 8.

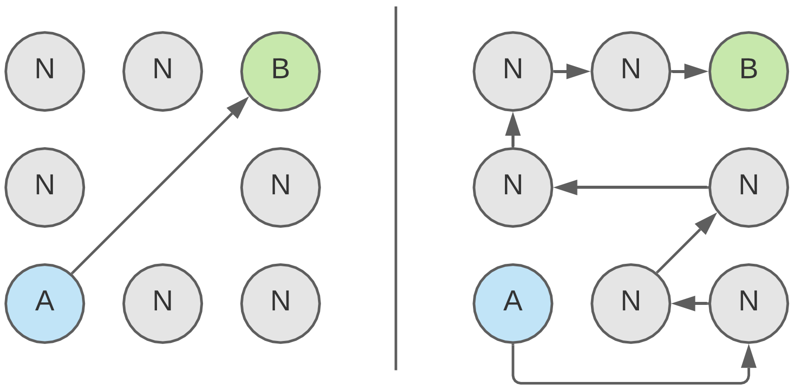

Figure 8: A Whisper network with zero and complete darkness, respectively.

Because of probabilistic message propagation, the time it takes for a message to reach the node it is intended for is random, but the average waiting time increases with each new node in the network. Note that even though an encrypted message is sent to every node, only the user whom it was intended for can decipher it. While the level of darkness can be modified to reduce latency, it always comes at a cost of reduced privacy.

3.2.2 Time to Live

Whisper is asynchronous, meaning that users who are eligible to receive a message do not have to be online at the time of its submission. While there is no permanent storage of content, messages have a modifiable Time-To-Live (TTL) property that dictates for how long it should exist on the network. Data that needs to exist for a long period of time should be stored in a smart contract on the blockchain, but transient data is perfectly suited for Whisper.

3.2.3 Denial-of-Service Attacks

One of the main concerns for any network with instant payload transmission is Denial-of-Service-attacks (DoS) where a foreign entity bombards a network with a massive number of packages within a short time span. This type of attack can quickly cause a network to overload and stop processing legitimate packages, particularly when the malicious packages are intentionally manipulated to take longer for the network to process. (Carl et al. 2006)

It sounds bizarre to want to build a communication protocol with properties like darkness and TTL while knowing how much damage a properly coordinated DoS attack could do to it, because to an extent it is. Whisper circumvents this problem by forcing users to solve a Proof of Work (PoW) puzzle of sufficient complexity before their message is propagated to the network. The puzzle's complexity is directly correlated with the message's size and TTL property, so increasing either will result in a significantly more difficult computation. Network peers are also allowed to reject messages that do not fulfill a minimum PoW threshold on an individual and topic level, so worthless spam can be filtered out.

4 SMART CONTRACT DESIGN PATTERNS

Smart contracts can feel quite limiting at times because many features and functionalities that are taken for granted in other programming languages are not built into the default suite. This means that developers often have to get creative to figure out alternative methods and design patterns to solve a particular problem.

This section will cover two prominent design patterns that expanded the horizon of what is accomplishable with smart contracts. Note that these patterns are more like soft guidelines rather than fully exhausted patterns and that the smart contract community is still collectively developing them.

4.1 Oracles

Normally smart contracts only have access to information that exists on the blockchain, but by utilizing a particular design pattern a smart contract can function as a gateway between the internal and external world. Smart contracts that serve such a purpose have been named oracles and usually have a supportive role in a larger system, rather than being the primary objective of it.

Ironically, smart contracts are not all that smart on their own and need access to external information to have real-world application and to be truly considered as an alternative to existing platforms. However, injecting external information like financial data through an API to the blockchain environment directly infringes upon its decentralization because the data is often owned by a central authority. This conflict is referred to as the “oracle problem” and is the sole focus for many researchers because the solution is Ethereum’s proverbial golden ticket to mainstream adoption.

4.1.1 Event Loop

Smart contracts have built-in event listeners (called events) that can be subscribed to for asynchronous notifications of variable changes or function executions. Oracles use function events as a process trigger that a program that exists outside of the blockchain ecosystem is listening to. After detecting an event, the software runs the event parameters through an interpreter and promptly performs the appropriate off-chain action that was requested before injecting the result into a smart contract. Figure 9 shows a visual representation of this process.

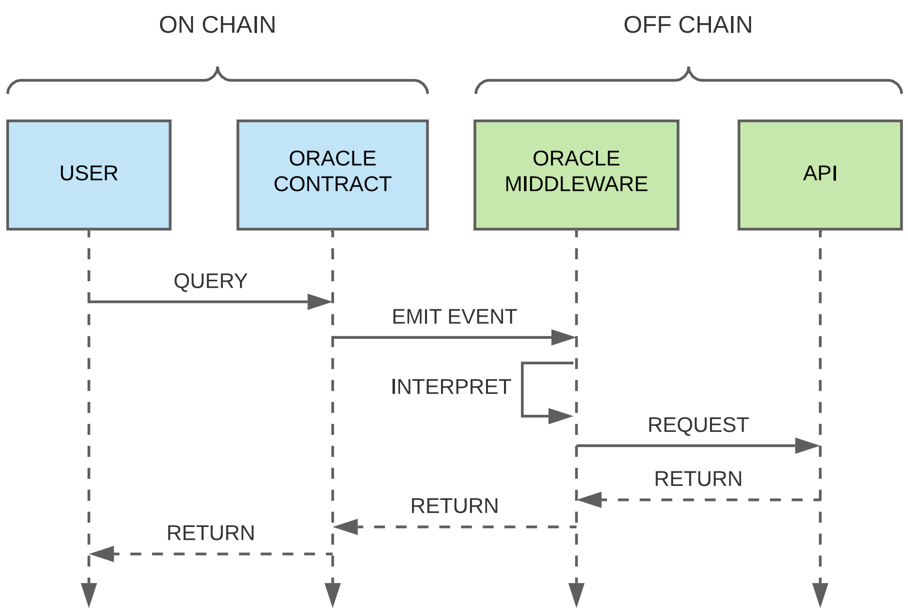

Figure 9: A smart contract oracle's action sequence.

4.1.2 Vulnerabilities

What an oracle's off-chain component can do is limitless, but an equally important question is how it does it. For example, the logistically simplest method of injecting financial data onto the blockchain is to request and forward it from a single source through an oracle, but that introduces multiple potentially catastrophic scenarios.

If a smart contract relies on data fetched from a single external endpoint to decide how its internal funds should be distributed, someone only needs to compromise that endpoint to manipulate the outcome in their favor. The solution is to diversify the result by requesting the same data from multiple sources and averaging the response. This obviously does not remove the possibility that someone could hack each API simultaneously, but significantly reduces the probability of it. Many machine learning models are also good at detecting outliers in data which could serve as an extra layer of defense to prevent irreversible damage to a smart contract.

4.2 Tokenization

An Ether component can be attached to a smart contract function by using the keyword "payable" in its declaration. Ether trading is possible both on a user-to-user basis, as well as on a user-to-contract and contract-to-user basis. This means that smart contract services can be built to continuously generate revenue through an autonomous pay-per-play scheme.

While paying for relatively cheap microservices with Ether is entirely possible because of its seven denominations, it quickly becomes tedious to manually deal with Ether. The community’s solution for this is to take monetization to the next level with a design pattern called tokenization. A token contract allows applications to create their own internal microeconomy by letting users purchase tokens for Ether that can then be used as payment for services or traded with other users. This design pattern incentivizes the continued use of an application, and also allows investors to receive something tangible in return for their capital.

4.2.1 ERC standards

The popularity of token economies grew so rapidly that standardization had no chance to take root, which meant that almost every token looked and worked differently and the very thing that was intended to make payment processing easier, ended up making it more confusing. To alleviate this issue, the smart contract community later came together and collectively created a token standard that any token that wanted to be officially recognized needed to adhere to. The Ethereum Request for Comments (ERC) standard was proposed in November of 2015 and is still the de facto standard of today. (Ethereum Improvement Proposals, 2020)

The core principles for an ERC token are the same as for a cryptocurrency. To prevent endless inflation through minting, a maximum token capacity is established when the contract is created. After that capacity is reached, the only method of obtaining tokens is through trading with users. Traditional token contracts only release tokens through purchases, but a recently popularized addition also lets users earn interest for the tokens they are holding. Every ERC token has a name, symbol, and decimal denomination on top of the static functions that enable purchasing, selling, trading, and verification. Structurally, every ERC token contract must be identical as to allow for cross-token trading through a uniform API.

4.2.2 Initial coin offering

Crowdfunding is an increasingly popular method for organizations to get initial funding for an idea. An initial coin offering (ICO) is a form of crowdfunding where interested investors can purchase an application's tokens before it goes public, in hopes of the tokens increasing in value afterward. Smart contracts are particularly well designed for exactly this type of service.

5 INTERNET OF THINGS

Internet of things (IoT) are devices that often perform a singular task as part of a network of similar machines, which communicate through the Internet to solve a problem collectively. These devices can generally be divided into two categories: microprocessors and microcontrollers. A microprocessor only has a central processing unit (CPU) and is used as an embedded system in a separate machine. A microcontroller is more like a standalone machine with a CPU, memory, hard drive, and an operating system.

IoT devices can be found anywhere from remotely controlled home appliances to automated factories to self-driving cars. They have already been used to aid in the management of large metropolitan cities (Smartdubai.ae, 2016). The true value of IoT can be realized through machine learning and real-time analysis of diverse data, but numerous obstacles must be overcome before that can happen.

5.1 Vulnerabilities

IoT is a billion-dollar industry that is riddled with privacy and security issues that are reminiscent of the Internet in the late 1990s and early 2000s. Services like Shodan (Shodan.io, 2013) prove that a massive number of owners do not practice proper password conventions or follow even rudimentary security procedures to protect their devices. To put it bluntly, IoT is a hacker's paradise in its current state.

Many devices ship with overly permissive default settings and simple ways to disable security layers to make the initial deployment of the device as smooth as possible. A consequence of this is that network ports that would normally be protected are now exposed. In the worst-case scenario, a device's entire operating system can be accessed through one of these vulnerable network ports. IoT devices are often interconnected with other devices, so one breached device works as a gateway to an entire network of other devices. The biggest problem in these scenarios is that the owner of the compromised device has almost no way of identifying breaches unless the hacker explicitly wants to make their presence known.

5.2 Device Hardening

Pre-emptive protective measures for systems are often referred to as "hardenings" and can address internal and external threats. Internal threats are system attacks done via software, and external threats are physically imposed attacks against a system's hardware. Most systems can afford to exclusively focus on internal threats because the machine is physically locked away in a safe place somewhere. IoT devices seldomly have this luxury and have to deal with the arguably greatest security threat of them all, hackers having physical access to the device. Chipsets can be soldered directly onto a motherboard to override its operating system's managerial capability or monitor its activity. There is very little that can be done if a device is physically attacked, but there should at the very least be automated methods of detecting physical changes in the device's hardware built into any software that it runs.

Not only is the detection of vulnerabilities difficult, but remotely updating device firmware and software is also tedious, particularly when done via secure encrypted channels. However, since IoT devices tend to serve a very specific purpose, their entire workflow is relatively easy to automate. A device that autonomously performs actions according to a predetermined set of rules is significantly easier to manage because problems can be solved in a virtual environment that mimics the behavior of the device. Once a fix is found, it can be neatly packaged and uploaded to a location where the device is preprogrammed to look for updates. After downloading the package, the device then replaces its old software with the new one. This allows the owner to harden the device against internal threats by completely disabling incoming connections, which significantly reduces the number of attack vectors for any potential hacker.

5.3 Revenue Streams

There is an undeniable market for anonymous data that is produced by different kinds of devices. Modern manufacturers are rarely willing to take risks with new products and instead base their entire assortment on their clientele’s consumption habits. Having access to large quantities of niche data is also the lifeblood of machine learning. The way companies tend to obtain this type of data is to simply take it from clients because there is a legal clause buried deep within the end-user license agreement (EULA) or a similar legal document (Kumar et al. 2019). Another problem is that there is no popular marketplace that would connect interested buyers with device owners. With little to no precedents of prior exchanges, negotiations about the reward and validation of data would demand significant effort to resolve.

Computation can also be sold as a commodity. Microcontrollers and smartphones are not particularly powerful machines, but they can be harnessed through horizontal scaling to solve very complex problems. To horizontally scale a computational problem essentially means that one unmanageable problem is divided into potentially thousands of small problems that are distributed to an equivalent number of devices to be solved. Raj et al. (2020) concluded that horizontal scaling via deceptively weak devices can quickly become a more cost-effective method of solving computationally intense problems compared to state-of-the-art hardware clusters that are being sold on the market today.

5.4 Projected Future

In machine learning, the accuracy of a prediction of future events is highly correlated to the quantity and diversity of the data that the model takes into consideration. A general rule of thumb is, the more data, the better the result. The same pattern can be applied to the IoT sector where autonomous devices measure something and predict whether an action (like a sprinkler) must be triggered or not. Basing that decision on one metric alone is irresponsible and bound to trigger significantly more false alarms due to a malfunction or misreading compared to a multi-metric prediction. This methodology simultaneously produces statistically superior predictions and makes the entire process more robust and difficult to manipulate with malicious intent.

Due to these insights, it is easy to understand why specialists think that the amount of IoT devices in the world will grow so much over the next few years (Newman, 2020). Anyone who wants to maximize the predictability of something will want to monitor and collect data about everything they possibly can. Even something as trivial as monitoring the setting of a home could require potentially thousands of devices and sensors.

6 IMPLEMENTATION

This section will showcase how the principles of the earlier sections can be applied in a practical setting by creating three distinctly different components. The system that is shown in Figure 10 simplifies the creation, management, and usage of smart contract oracles on the Ethereum blockchain. This ordeal requires multiple different subsystems and modules to coexist and communicate to collectively form a cohesive platform.

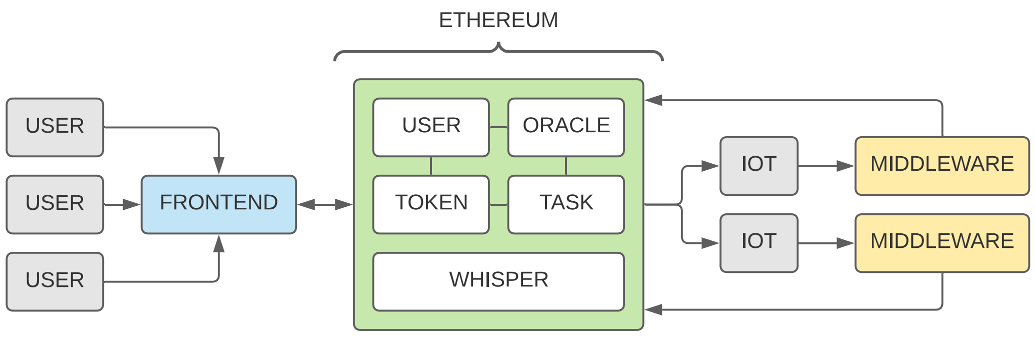

Figure 10: The proof-of-concept project visualized.

The distributed components will be analyzed on a relatively high level to highlight the subtleties in the architecture, but the detailed source code can be viewed directly from the respective components GitHub repository:

Smart contract backend:	https://github.com/wickstjo/oracle-manager
IoT middleware:	https://github.com/wickstjo/iot-manager
Frontend application:	https://github.com/wickstjo/distributed-task-manager

6.1 Component Overview

The backend system consists of four interlinked smart contracts. Ethereum users will be able to register themselves as system users that can then publish and monetize smart contract oracles. An oracle's owner is authorized to modify an oracle's configuration, but other users are also permitted to command the device to perform specific actions by creating tasks. The platform will utilize its own token economy by incorporating a two-way token staking model for instantiated tasks.

To automate an oracle's workflow and enable indirect management of a device, we will create a Python interpreter that will henceforth be referred to as the "middleware". The middleware receives events from the oracle's unique smart contract that it then interprets and uses to update its current state, which effectively allows it to function autonomously. To enable dynamic discovery of oracle services, all middleware instances will also be connected to the same whisper topic. When a discovery request is received, the middleware will compare its unique configuration to the stipulations of the request and respond when a match is found.

Finally, we will create a frontend application that streamlines the usage of the backend system through a graphical interface in the form of a web application. The middleware will also have an external component called the "oracle client" that allows users to detail the objective of their oracle device.

6.2 Development Tools

6.2.1 NodeJS

NodeJS is a cross-platform JavaScript runtime environment with a built-in package manager called NPM. We will be using the 14.15 version of NodeJS with the following libraries and frameworks:

1. **Create React App** – A JavaScript framework for frontend applications.

2. **Web3 JS** – A collection of libraries that allows the frontend application to communicate and interact with Ethereum’s protocols.
3. **Truffle** – A smart contract development framework that can be used to compile, unit test, and deploy contracts to the blockchain.
4. **Ganache** – An in-memory development blockchain.

6.2.2 Python

Python is a high-level programming language that most machines and devices can interpret and execute with factory settings. It has a package installer called PIP. We will be using the 3.6 version of Python with the following libraries:

1. **Jupyter Notebooks** – Notebooks are an excellent development environment where blocks of code can be executed in segments rather than all at once.
2. **Web3 PY** – A library that allows communication and interaction with Ethereum’s protocols.

6.2.3 Geth

Go Ethereum (Geth) is a command-line interface that functions as the entry point to any Ethereum network. With Geth, users can synchronize and interact with a blockchain but also create new private blockchains and enable their machine to function as a gateway for others. We will be using the 1.9 version of Geth.

6.3 Smart Contract Backend

A distributed smart contract backend system works very similarly to a traditional backend system that uses relational databases. To efficiently interact with a database, modern applications need to build a separate application programming interface (API) to handle requests from the frontend. A smart contract is essentially a complex database with a built-in API that dictates how it can be interacted with and by whom.

The project’s backend system consists of multiple individual smart contracts that are bound together by injecting immutable references to each contract in a process called initialization. Initialization can only be performed once, which after the contract becomes locked and further initialization attempts will be rejected by the Ethereum virtual machine. We will repeatedly use a factory contract design pattern where a manager contract oversees the creation of new and indexing of old child contracts of that type.

Four types of manager contracts deal with different aspects of the backend system. They can (and do) query each other for information when deciding if an operation in its jurisdiction is allowed to be executed or not. After being initialized, manager contracts become autonomous and only rely on each other for validation. At no point can a human intervene in a process and force a smart contract to do something that its internal logic does not allow. Figure 11 presents the variables and functions of each manager contract and their respective child contracts.

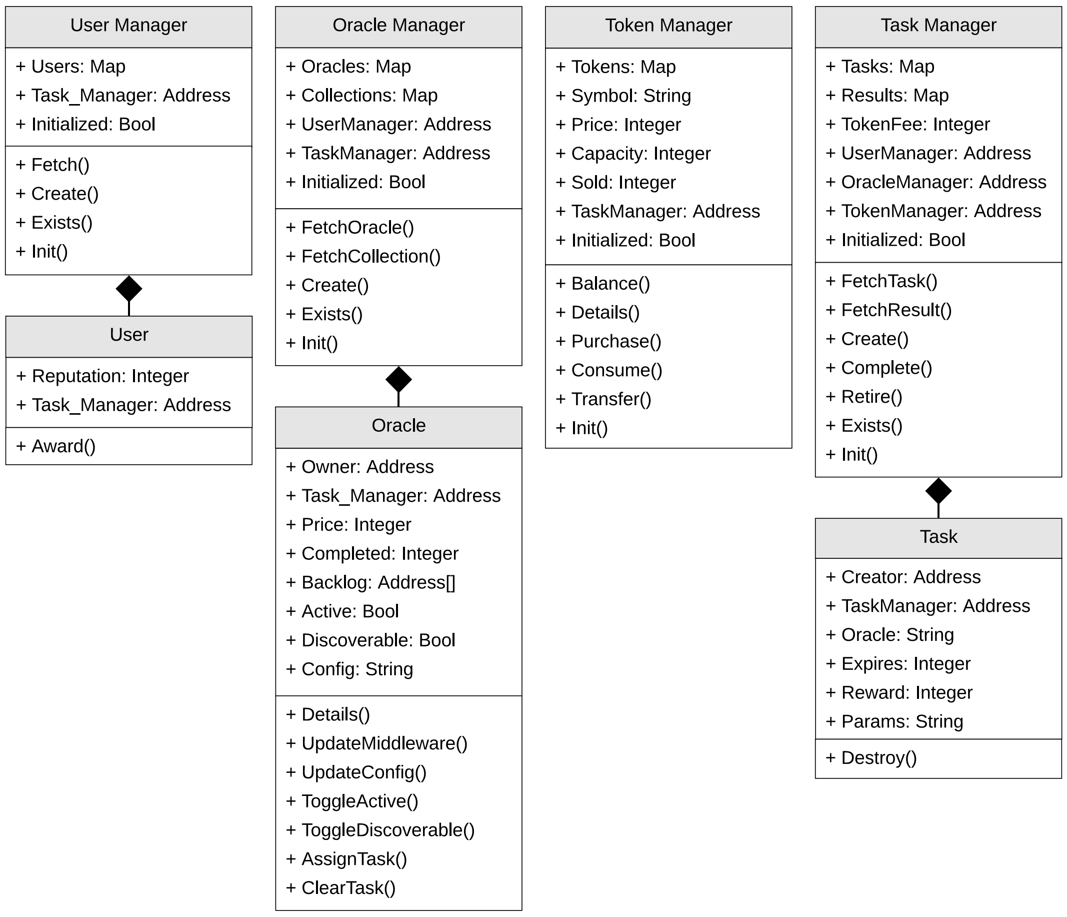

Figure 11: The variables and methods of the smart contract backend system.

A distributed backend allows two parties, human or machine, to communicate indirectly through a common API which opens interesting possibilities particularly for the IoT sector where direct communication is the source of many of its vulnerabilities.

6.3.1 User Manager

The user manager and its children are the least complex smart contracts in this system. Registered users are indexed in a hash map defined data structure, where the public key of an Ethereum account serves as the key and the instantiated user contract as its value. The only requirement for registration is that the public key cannot already be registered since it would overwrite an existing contract. The user contract is equally uneventful and only has a reputation variable that represents how many cumulative tasks have been completed by the user's oracles. Only the task manager has permission to modify the reputation variable of a user via the award function.

Both the user manager contract and its child contracts can seem almost redundant with how little functionality they offer to the other manager contracts. However, forcing users to register before they have permission to interact with the other manager contracts is an easy way to slightly increase the trustworthiness of a system that is inherently untrustworthy. The reputation variable is another method of achieving a similar effect. It gives users a quick and easy method of checking other users' previous commitments to make a more informed decision whether they are trustworthy enough for a particular task.

6.3.2 Oracle Manager

The oracle manager has a similar structure and function as the user manager, but oracle creation requires two extra parameters, and the indexing logic is expanded slightly. The first of these two parameters determine how many tokens the oracle should charge for its services, and the second parameter is a unique identification string that will be used for indexing and therefore required for fetching.

The second parameter can technically be any arbitrary string that has not already been used, but since we want oracles to be able to function autonomously, this identifier needs to be generated deterministically. One solution for this is to fill in a standardized IoT form (Hinterberger et al. 2020) and pass it through a SHA256 hash function. As long as the form is stored locally on the device, the hashing process can be effortlessly repeated and fulfills the criteria. This method is not guaranteed to always produce an unreserved identification hash, but the probability of multiple users filling in an identical form is negligible. In scenarios where a conflict does occur, the second user can slightly modify the content of the form and drastically change the resulting hash.

Every created oracle is stored in a hash map data structure (named oracles) with the hash identifier as the key and the oracle contract as its value. However, every oracle is also indexed in another hash map (named collections) where the key is a user, and the value is a list of oracles that are linked to the user. The former hash map is used by manager contracts to quickly fetch and verify an existing oracle, and the latter provides a user-friendly way of assembling a specific user's oracle collection without needing to remember the device identifiers.

An oracle contract contains variables that dictate its middleware's behavior, so the functions with the capability of modifying these values must be restricted to the owner. The owner is allowed to toggle the active and discoverable status, change how other users can discover the oracle by updating the discovery configuration, and even trigger middleware updates. Any modification to an oracle's smart contract is automatically emitted to its middleware through the sequence of actions that are presented in Figure 12. This is a form of indirect management of the oracle device.

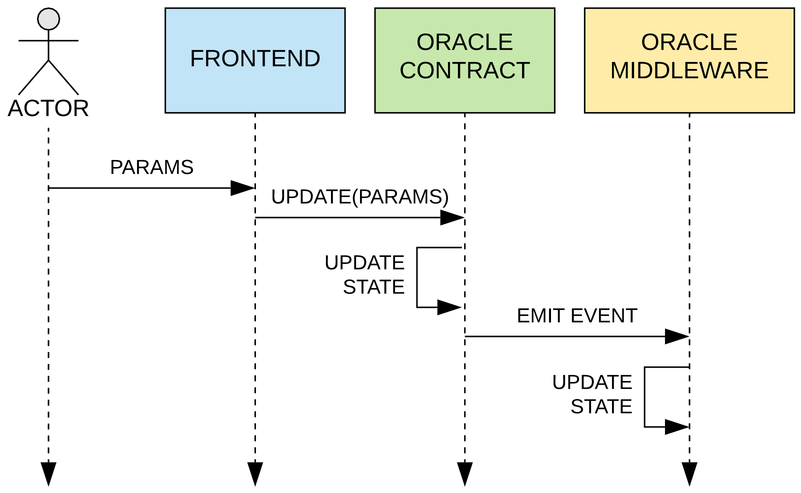

Figure 12: The owner updates an oracle's configuration.

The backlog is a list of incomplete tasks that the oracle has been assigned. To enforce that proper procedure is always followed in matters with multiple parties, only the task manager has permission to assign and clear tasks from an oracles backlog. The middleware executes tasks one-by-one according to the first-in-first-out principle and returns results to the task manager for verification.

6.3.3 Task Manager

The task manager is the central pillar in the backend system because it has jurisdiction over certain aspects of contracts other than itself. There is at least one function in each contract that only the task manager is permitted to execute, but a smart contract cannot autonomously initiate a transaction. However, if a task manager function calls a sub-function that is restricted to the task manager, then the EVM interprets that as the task manager being the sender and the criteria is met. In other words, these restricted functions are only indirectly executable with the task manager's consent. Figure 13 presents the sequence of actions that occur when a new task is created. Note that task instantiation, token seizing, and task assignment are task manager restricted functions that are permitted to execute because the actor calls them indirectly.

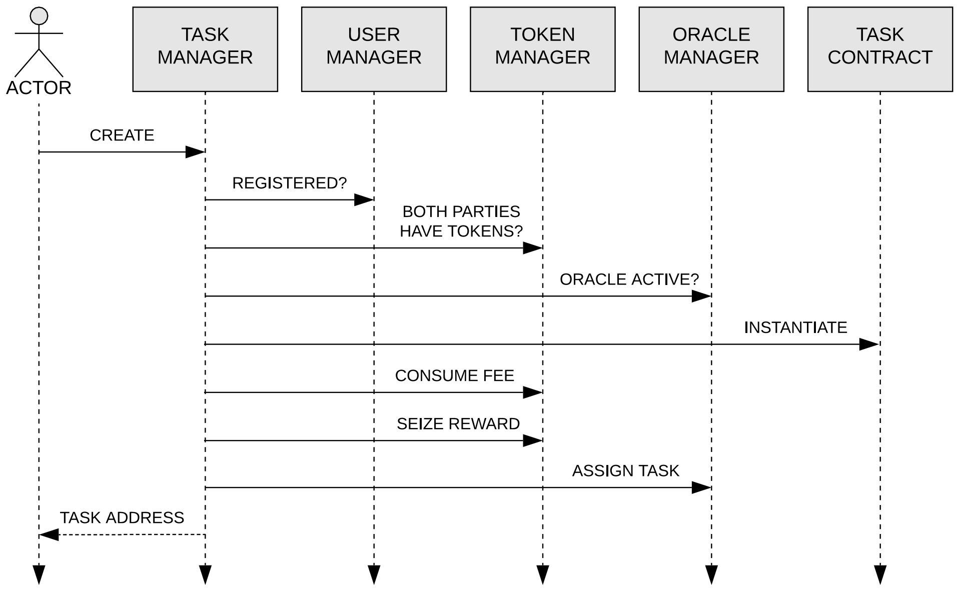

Figure 13: The sequence of events when a user creates a new task.

A task is a formal and binding contract between a user and an oracle where the user requests an oracle to perform an action for a token reward. Task creation, completion, retirement, and every other restricted function is triggered by human-users but overseen by the task manager. This is significant because of the initialization process that was described at the beginning of section 6.3. Users are safe to assume the worst about each other because there is an incorruptible autonomous intermediary between them to oversee that everything goes according to a predetermined plan and that any straying from it will be mercilessly penalized.

Tasks are transient by definition and adhere to a specific life cycle which contradicts the permanent nature of smart contracts. After a task has served its purpose by either being completed or its deadline expires, it no longer needs to exist. In smart contract terms, this signifies self-destruction, which invalidates the contract by setting every variable and function it contains to null. After a task is completed or retired, the task manager automatically self-destructs the task contract and removes it from the oracles backlog. Note that self-destruction is not synonymous with deletion and that a destroyed task contract's original state can still be inspected from the block history.

To incentivize fairness, the oracle is also required to stake tokens before a task can be assigned to it. When both parties have something to lose, an economic model can be built where one party is always incentivized to see a task through instead of abandoning it. This also means that under normal circumstances, the gas cost for a task's life cycle is split relatively equally. One party pays for creating the task, and the other for completing it. If an oracle is unable to complete a task before its deadline, the creator can retire the task and be compensated with the entire token pot instead.

6.3.4 Token Manager

The token manager is a modified ERC-20 token contract that demonstrates how a token economy could be envisioned and built around oracles or any other repeated service that exists on the blockchain or otherwise.

The staking model for tasks works with the following ruleset:

1. The task creator must stake at least as many tokens for a task as the oracle's service price variable dictates.
2. Creating new tasks consumes one token from the creator.
3. To accept a task, an oracle must first stake half the reward in tokens.
4. The task manager seizes all staked tokens until the task is destroyed.

It is important that the oracles token stake is correlated to the reward because it gives users a method of specifying the urgency or importance of a task by increasing the reward. By default, the middleware executes tasks in chronological order, but it would be relatively easy to reconfigure to prioritize tasks with a higher reward, similarly to how increasing a transaction's gas property will get it mined quicker. The current stake percentage was chosen completely arbitrarily but should likely be dynamically generated based on the tasks reward in a production scenario.

Token sales are how the developers of a smart contract system profit monetarily over time. By consuming a small number of tokens every time a task is created, those tokens can be sold on the marketplace again without ever exceeding the maximum capacity. Most tokens do not have a method of circulating and selling old tokens and instead use a dynamic price, but this model promotes hoarding over spending which can be detrimental to the application's internal economy.

6.4 Middleware

The middleware is the off-chain component of an oracle that oversees interpretation. It receives events from the oracle's smart contract for sensitive matters that require authority, but also from the Whisper protocol in matters regarding service discovery. Since internal hardening significantly increases the trustworthiness of a machine, we want to encourage users to conform to it without making it mandatory. To make the barrier to entry as low as possible, the middleware will also handle outgoing communications and inherit functions from a completely separate and customized Python file called the "oracle client" that will be explored in subsection 6.4.3.

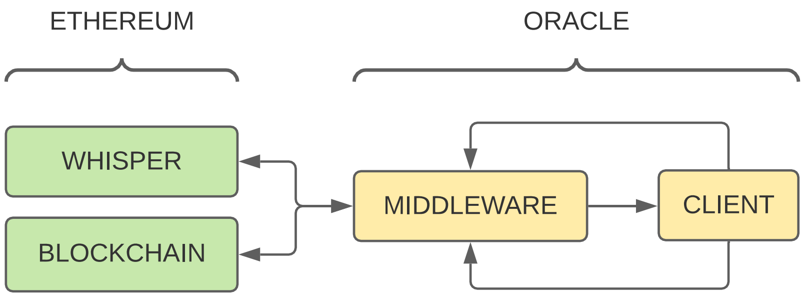

Figure 14: The incoming and outgoing dataflow of the oracle middleware.

Note that the middleware implementation shown in Figure 14 is only a suggestion of how the usage of a smart contract backend could be automated and is in no way a requirement. The Ethereum blockchain's distributed nature means that a smart contract's Application Programming Interface (API) cannot be hidden and is in fact completely transparent and open source. Anyone is permitted to build their own variation of an interpreter and use a smart contract to its full potential with it.

6.4.1 Configuration

The middleware is dynamic by design and modifies its behavior when something in its smart contract changes. The owner can apply sweeping changes like disabling task assignments to an oracle by toggling a Boolean value, but also intricate changes like what type of discovery requests the middleware should respond to by modifying the discovery parameters in the configuration variable.

An oracle has two types of properties, static and dynamic. Static properties should be provided and used in the identification process described in subsection 6.3.2, and the rest should be included in the dynamic configuration. On the device, the configuration is a JSON object but because smart contracts have no similar data structures, the object is put through a base64 transcoder and saved as a string in the smart contract. When an oracle's device launches its middleware, the encoded configuration is fetched from the smart contract and serialized back to a JSON object before it can be used.

6.4.2 Service Discovery

The project's service discovery system is built using a Whisper topic that all oracles are connected to. Requests and responses are both base64 transcoded JSON objects with a header and message property that changes depending on the category. When the middleware intercepts a service request, it compares its dynamic configuration to the stipulations of the request and responds with an availability message if a match is found.

While a service discovery system could technically be built using smart contracts, it would be expensive and slow to use. The Whisper protocol is intended for short and snappy communication between machines and costs absolutely nothing to use, so we only needed to design a publish/subscribe pattern that fits our use case.

6.4.3 Oracle Client

The oracle client is an independent Python program with a set amount of base functions that the middleware inherits and executes upon receiving task assignments. By detaching these components, the owner can focus on declaring the oracles actions in the software while the middleware takes care of event handling and communication.

Task contracts must be structurally homogenous even though an oracle can serve almost any purpose or perform any action. This contradiction is overcome by transcoding a JSON object with the task's hyperparameters into a string and saving it in the task contract. When a task is intercepted by the middleware, the hyperparameter string is automatically decoded and passed as an argument to the inherited function. Figure 15 is a continuation of Figure 13 and shows the complete life cycle of a task.

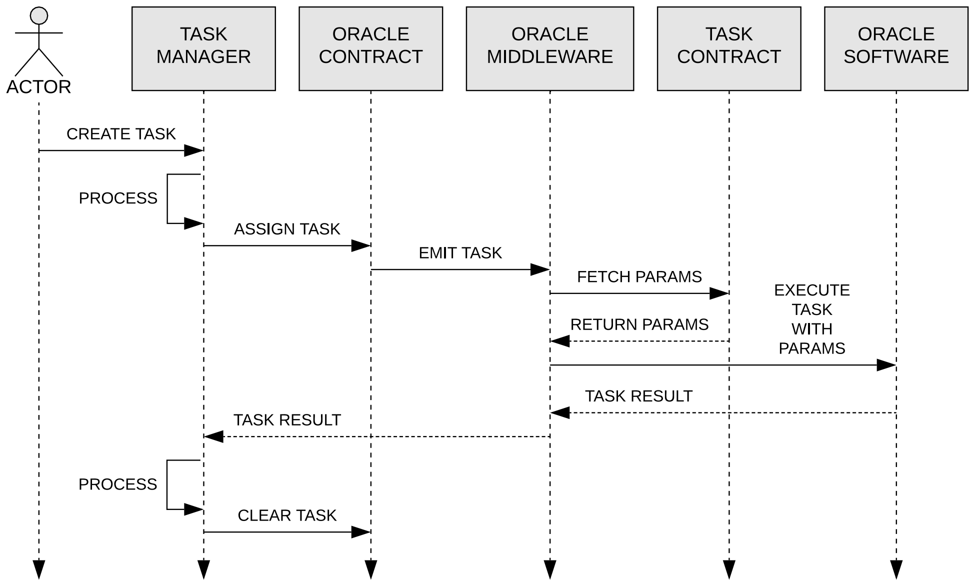

Figure 15: A task's complete life cycle.

The benefits of this are three-fold. Firstly, it allows for a limitless number of diverse hyperparameters without penalizing oracles that require few or many. Secondly, permanent instructions are necessary for the result verification process when checking if an oracle has completed a task correctly. Thirdly, this is user friendly because complex machine instructions can be written in a structured file format like JSON or YAML that can then be dropped into the frontend application for further parsing.

6.5 Frontend Application

6.5.1 React Framework

The frontend application implementation uses a JavaScript framework called React that is developed and maintained by Facebook (Reactjs.org, 2019). While the framework was initially only intended to be used for web development, other developers have ported its functionality and effectively created daughter frameworks that compile react code into mobile, tablet, and desktop applications. Many languages and frameworks can create applications that run slightly more efficiently, but none of them allow for essentially the same codebase to be rendered on all devices and platforms simultaneously like React does.

6.5.2 Distributed Applications

A traditional backend system is locked behind an authentication barrier, and the only method of interacting with them is through a frontend application that was created by someone with credentials. The frontend application is usually hosted on a server and made available to the public under a particular IP address or domain.

This paradigm changes completely when the backend system is a distributed smart contract that does not have to be similarly protected. Just like the middleware implementation, the frontend application is merely another optional method of streamlining the usage of the smart contract backend. If an “officially released” application does not suit a particular use case, anyone can create a separate application and use it in a production environment instead, without being penalized.

Distributed web applications do not need to be tied to a particular domain because a locally hosted web application works the same way as a globally hosted one does. While a globally hosted web application is convenient to use, the central location also makes it easier for governments and other authorities to block its content for a wide assortment of users. Local applications are not affected by traditional issues like cross-site-scripting or session hijacking, and only require a reference to a gateway through which the blockchain can be accessed safely.

7 CONCLUSION

This work shows that the Ethereum platform can indeed be utilized to neutralize many of the vulnerabilities that plague Internet of Things (IoT) by replacing trust with autonomous systems and algorithms. While this dichotomy does change certain architectural paradigms, distributed components can be used in a relatively familiar fashion and implemented incrementally over time. Managing IoT devices through a blockchain significantly reduces the number of attack vectors that can be used to impair a device, but it also increases the cost of use and latency.

The components of section 6 give some insight into the design process of a decentralized application, but the scope of the project is perhaps too ambitious. Smart contracts are inherently bad at nuanced problems, so conditional statements must be formulated to have a binary label. The included project outlines a platform for ambiguous oracles, which introduces a certain amount of unavoidable nuance to the equation. As a result, the validation process for tasks is extremely complex and remains largely undeveloped and insufficiently researched. By introducing more constraints to what an oracle can do or what a task can be, the project would become remarkably easier to implement and safer to use.

It should also be understood that blockchain technology is still in its infancy and remains untested on a grand scale. The recent proliferation of IoT has also been so significant that it is impossible to predict what the future infrastructure will have to endure. Regardless, both technologies show great promises for various use cases. The Ethereum Foundation recently reached its staking goal and were able to launch the blockchain protocols 2.0 version (Ethereum Foundation, 2020). It directly addresses many prominent scalability issues for transactions and smart contracts alike. Additionally, the Swarm protocol (Trón et al. 2020) that intends to solve distributed storage, which is a crucial component for distributed applications, has recently been made available to the public. Consequentially, solutions related to Ethereum keep growing stronger.

REFERENCES

Nakamoto, S., 2008. *Bitcoin: A peer-to-peer electronic cash system.*

Buterin, V., 2014. *A next-generation smart contract and decentralized application platform*. white paper, 3(37).

Shodan.io. (2013). *The world's first search engine for Internet-connected devices.* [online] Available at: https://www.shodan.io/.

Ethereum Foundation. (2020). ethereum/eth2.0-specs. [online] Available at: https://github.com/ethereum/eth2.0-specs.

docs.soliditylang.org. (2020). Solidity 0.7.5 documentation. [online] Available at: https://docs.soliditylang.org/ [Accessed 3 Dec. 2020].

Raspberry Pi Foundation. (2019). *Raspberry Pi — Teach, Learn, and Make with Raspberry Pi*. [online] Raspberry Pi. Available at: https://www.raspberrypi.org/.

Garfinkel, S., 1995. PGP: pretty good privacy. " O'Reilly Media, Inc.".

Truong, Nguyen & Jayasinghe, Upul & Um, Tai-Won & Lee, Gyu Myoung. (2016). *A Survey on Trust Computation in the Internet of Things*. THE JOURNAL OF KOREAN INSTITUTE OF COMMUNICATIONS AND INFORMATION SCIENCES (J-KICS). 33. 10-27.

Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "*An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends*," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 557-564, doi: 10.1109/BigDataCongress.2017.85.

Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. *On the Security and Performance of Proof of Work Blockchains*. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 3–16. DOI:https://doi.org/10.1145/2976749.2978341 cbeci.org. (2020). *Cambridge Bitcoin Electricity Consumption Index* (CBECI). [online] Available at: https://cbeci.org/ [Accessed 3 Dec. 2020].

Li W., Andreina S., Bohli JM., Karame G. (2017) *Securing Proof-of-Stake Blockchain Protocols. In: Garcia-Alfaro J., Navarro-Arribas G., Hartenstein H., Herrera-Joancomartí J. (eds) Data Privacy Management, Cryptocurrencies and Blockchain Technology*. DPM 2017, CBT 2017. Lecture Notes in Computer Science, vol 10436. Springer, Cham. https://doi.org/10.1007/978-3-319-67816-0_17

Ethereum Wiki. (2020). *whisper-overview.* [online] Available at: https://eth.wiki/concepts/whisper/whisper-overview [Accessed 3 Dec. 2020].

Shinde, S.A., Nimkar, P.A., Singh, S.P., Salpe, V.D. and Jadhav, Y.R., 2016. *MQTT-message queuing telemetry transport protocol.* International Journal of Research, 3(3), pp.240-244.

Trón, V., Fischer Á., Johnson N., Nagy D., Felfö Z. (2020). *The book of Swarm.* [online] Available at: https://readthedocs.org/projects/swarm-guide/downloads/pdf/latest/ [Accessed 3 Dec. 2020].

Pouwelse, J., Garbacki, P., Epema, D. and Sips, H., 2005, *February. The bittorrent p2p file-sharing system: Measurements and analysis. In International Workshop on Peer-to-Peer Systems* (pp. 205-216). Springer, Berlin, Heidelberg.

Wuille, P., 2012. Bip32: Hierarchical deterministic wallets. h ttps://github. com/genjix/bips/blob/master/bip-0032. md.

Metamask.io. (2019). MetaMask. [online] Available at: https://metamask.io/.

Git-scm.com. (2019). Git - Documentation. [online] Available at: https://git-scm.com/doc.

mythx.io. (2020). *MythX: Smart contract security service for Ethereum.* [online] Available at: https://mythx.io/ [Accessed 3 Dec. 2020].

swcregistry.io. (2020). *Smart Contract Weakness Classification and Test Cases.* [online] Available at: https://swcregistry.io/ [Accessed 3 Dec. 2020].

Etherscan.io (2019). *Ethereum (ETH) Blockchain Explorer*. [online] Ethereum (ETH) Blockchain Explorer. Available at: https://etherscan.io/.

Ethgasstation.info. (2016). *ETH Gas Station.* [online] Available at: https://ethgasstation.info/.

Cloudflare. (2020). *Ethereum Gateway - Cloudflare Distributed Web Gateway docs.* [online] Available at: https://developers.cloudflare.com/distributed-web/ethereum-gateway [Accessed 3 Dec. 2020].

Infura. (2020*). Ethereum API | IPFS API Gateway | ETH Nodes as a Service.* [online] Available at: https://infura.io/.

Oikarinen, J. and Reed, D., 1993. *Internet relay chat protocol.*

Carl, G., Kesidis, G., Brooks, R.R. and Rai, S., 2006. *Denial-of-service attack-detection techniques.* IEEE Internet computing, 10(1), pp.82-89.

Ethereum Improvement Proposals. (2020). *EIP 20: ERC-20 Token Standard.* [online] Available at: https://eips.ethereum.org/EIPS/eip-20.

Smartdubai.ae. (2016). [online] Available at: https://www.smartdubai.ae/.

Kumar, D., Shen, K., Case, B., University, S., Garg, D., Alperovich, G., Kuznetsov, D., Gupta, R. and Durumeric, Z. (2019). *All Things Considered: An Analysis of IoT Devices on Home Networks All Things Considered: An Analysis of IoT Devices on Home Net-works.* [online] Available at: https://www.usenix.org/system/files/sec19-kumar-deepak_0.pdf.

Raj, E., Westerlund, M. and Espinosa-Leal, L., 2020. *Reliable Fleet Analytics for Edge IoT Solutions*. CLOUD COMPUTING 2020, p.55.

Newman, P. (2020). *THE INTERNET OF THINGS 2020*. [online] Business Insider. Available at: https://www.businessinsider.com/internet-of-things-report?r=US&IR=T [Accessed 3 Dec. 2020].

Hinterberger, L., Weber, B., Fischer, S., Neubauer, K. and Hackenberg, R., 2020. IoT Device IdentificAtion and RecoGnition (IoTAG). CLOUD COMPUTING 2020, p.17.

Reactjs.org. (2019). React – A JavaScript library for building user interfaces. [online] Available at: https://reactjs.org/.

APPENDIX 1 SWEDISH SUMMARY

Introduktion och Syfte

Den konceptuella idén om en fullständigt elektronisk valuta har existerat sedan 1990-talet. Att åstadkomma förtroende för ett ting som inte är förankrat i det fysiska har genom tiderna varit det största problemet för elektroniska valutor. En forskningsrapport blev dock publicerad år 2008 som påstod att förtroende kunde ersättas med kryptografi (Nakamoto, 2008). Kort därefter utvecklades en blockkedja med namnet Bitcoin som bevisade att påståendet faktiskt var genomförbart och kunde tillämpas på en stor skala.

Syftet med detta examensarbete är att utforska blockkedjeteknologi, speciellt Ethereum-plattformen och dess protokoll, och att förstå olika säkerhetsbrister på den teknologiska marknaden. Efter en grundlig utforskning kommer dess principer att tillämpas i en praktisk implementation i form av distribuerade komponenter.

Blockkedjeteknologi

En blockkedja är en avancerad databas som indexerar transaktioner i länkade block. Databasen är inte ägd av en central entitet, utan består av ett helt nätverk av operatörer varav alla har tillgång till hela databasens innehåll. På grund av dessa orsaker så kallas en blockkedja ofta till en distribuerad plånbok. Distribuerat ägarskap är inte speciellt normalt inom mjukvaruutveckling för att det är väldigt svårt att bygga en fungerande praxis som alla nätverksnoder skall anpassa sig till. Blockkedjeteknologi överkommer den här problematiken med en så kallad ”konsensusalgoritm” som autonomt dirigerar och ser till att nätverkets användare uppför sig korrekt. Konsensusalgoritmen har en väldigt rigorös och sträng valideringsprocess för nya transaktioner. Godkända transaktioner är effektivt oföränderliga (eng. immutable) för att en retroaktiv modifikation av ett block också kräver en modifikation av alla senare block i kedjan. (Zheng et al. 2017)

Ethereum

I samband med Bitcoins uppkomst har utvecklare börjat experimentera hur blockkedjeteknologi kunde tillämpas för andra ändamål än penningtransaktioner. Ett prominent exempel av detta är Ethereum vars blockkedja kan exekvera maskinkod i form av

”smarta kontrakt”. Denna blockkedja bygger på samma principer som alla andra blockkedjor men har en expanderad arkitektur för att kunna stöda smarta kontrakt. Ethereum är egentligen en väldigt komplex tillståndsdator (eng. state machine) och dess blockkedja innehåller datorns hela tillståndshistorik. (Buterin, 2014)

Smarta Kontrakt

Smarta kontrakt kan skrivas i flera olika programmeringsspråk men är avsiktligt begränsade i sin funktionalitet för att undvika misstag och missförstånd. Blockkedjans oföränderliga natur angår också smarta kontrakt. Rent programmeringsmässigt betyder detta att nya variabler och funktioner inte kan skrivas in i ett kontrakt som redan har laddats upp till blockkedjan. Existerande variabler kan dock modifieras och dynamiska datastrukturer som listor och tabeller kan förlängas eller förkortas via funktioner som kallas ”setterfunktioner”. Smarta kontrakt använder ett versionshanteringssystem på ett liknande sätt som Git (Git-scm.com, 2019) vilket innebär att alla versioner av ett kontrakt sparas, men bara den senaste versionen visas åt användaren per automatik.

I och med att smarta kontrakt är oföränderliga medför också att logikfel i uppladdade kontrakt inte heller går att fixa retroaktivt. Detta innebär att det inte räcker med att bara implementera en viss funktionalitet som sedan provkörs i en produktionsmiljö. Alla försiktighetsåtgärder måste vidtas för att identifiera möjliga problem på förhand. Analysprocessen är väldigt omfattande men följer dock ett strukturerat mönster, vilket har lett till att flera företag försöker utveckla verktyg som skulle automatisera hela processen. Smarta kontrakts utvecklare uppehåller också en gemensam databas som innehåller alla kända sårbarheter som smarta kontrakt har. (Swcregistry.io, 2020)

Som tidigare påpekats är smarta kontrakt begränsade i sin funktionalitet, men detta skall inte uppfattas som om komplexa system inte ändå kunde byggas med dem. En av de mest restriktiva egenskaper är att smarta kontrakt inte har tillgång till information som inte redan existerar på blockkedjan. Kreativa utvecklare har lyckats kringgå även denna egenskap med ett ”orakel” designmönster som förvandlar ett smart kontrakt till en portgång mellan den externa och interna omgivningen. Restriktiva egenskaper existerar dock av god orsak, vilket innebär att kringgående designmönster också har oöverkomliga för- och nackdelar. Orakel är nödvändiga för att öka mängden användningsfall för smarta kontrakt, men de inför också en del centraliserade element till en teknologi som strävar till att vara helt distribuerad.

Whisper Protokollet

Ethereum-plattformen har flera protokoll som strävar till att lösa olika teknologiska problem med en distribuerad arkitektur. Blockkedjan är ett av protokollen och behandlar distribuerad beräkning (eng. computation) medan Whisper protokollet försöker lösa närkommunikation (eng. peer-to-peer communication). Whisper är helt identitetbaserad och byggd från grundnivån med säkerhet och anonymitet som dess högsta principer. Användare kan skicka asymmetriskt krypterade meddelanden till varandra, eller använda en gemensam symmetriskt krypterad kanal där alla deltagare kan dechiffrera inkommande meddelanden. (Ethereum Wiki, 2020)

Whisper ägs och drivs också av ett distribuerat nätverk av noder som styrs av en konsensusalgoritm. Inga meddelanden sparas beständigt på nätverket för att varje meddelanden har en justerbar livstid (eng. time-to-live). Alla skickade meddelanden sprids också sannolikhetsmässigt till alla nätverksnoder för att åstadkomma anonymitet. Detta beslut bygger på att ett specifikt meddelande inte skall gå att spåra tillbaka till en enda nod, och användare har därför plausibel förnekbarhet. Notera att fastän ett meddelande skickas till alla nätverksnoder så kan bara användare med tillgång till den rätta krypteringsnyckeln dechiffrera dess innehåll.

Sakernas Internet

Sakernas Internet (eng. Internet of Things, IoT) är apparater som ofta bara har ett specifikt uppdrag, men kommunicerar och jobbar tillsammans med andra liknande apparater för att kollektivt lösa ett problem. Dessa apparater kan hittas i allt från köksmaskiner till självkörande bilar och har redan tagits i bruk för att hjälpa styrandet av storstadsområden. IoT-industrin är beräknad att vara värd mer än 100 miljarder USD och specialister förutspår (Newman, 2020) att dess popularitet bara kommer att växa, men samtidigt är IoT-apparater också plågade av säkerhetsproblem.

Tillverkare har tendens att sälja apparater med alltför tillåtande standardinställningar och sätt att inaktivera säkerhetsåtgärder för att göra ibruktagandet möjligast lätt. Konsekvensen för detta fenomen är att flera nätverksportar som under normala omständigheter skulle vara skyddade är nu utsatta för hot. I värsta fall kan en apparats hela operativsystem vara åtkomligt via en öppen nätport. Ett av de få effektiva sätt att härda en apparat för att göra den mer uthållig mot attacker är att automatisera hela dess arbetsflöde och låta den arbeta autonomt. Denna metodologi tillåter ägaren att avvisa alla inkommande förbindelser vilket minskar en potentiell angripares attackvektorer betydligt.

Det skulle garanterat finnas en marknad för anonymt skickad IoT data med tanke på hur ovilliga moderna tillverkare är att basera sitt sortiment på något annat än deras klientels konsumtionsvanor. Fastän IoT-apparater inte är speciellt kraftiga förhållandevis till andra datorer, så kan de användas för att lösa väldigt komplexa beräkningsproblem via en vågrät skalningsarkitektur. En färsk studie konkluderade (Raj et al. 2020) att vågrät skalning med relativt svaga apparater kan snabbt bli ett mer kostnadseffektivt sätt att lösa beräkningsproblem jämfört med toppmoderna lösningar på den nuvarande marknaden.

Implementationen

Hypotesen bakom den praktiska implementationen är att Ethereum-plattformens ypperliga säkerhet torde kunna komplettera och förstärka sakernas internet (IoT). För att bevisa hypotesen så byggdes ett distribuerat system som förenklar skapandet av smarta kontrakts orakel. Detta krävde bland annat en backend-enhet i form av smarta kontrakt samt en mellanvara som körs på IoT-apparaten för att automatisera dess jobbflöde.

Backend-enheten består av fyra länkade smarta kontrakt som bara konsulterar varandra för att göra beslut. Ethereum-användare måste först registrera sitt konto för att sedan kunna registrera och publicera olika sorters orakel. Bara ägaren av ett orakel kan uppdatera dess konfiguration indirekt via dess smarta kontrakt. Andra användare kan dock också begära ett orakel att utföra en specifik uppgift genom att skapa ett uppdragskontrakt som delegeras till apparaten. För att förstärka användares förtroende måste båda parter av ett uppdrag avgå en pant som bara returneras efter uppdraget är fullgjort.

Mellanvaran tolkar händelser som sker i oraklets smarta kontrakt. När en variabel i kontraktet blir uppdaterad så avger processen också en signal som mellanvaran tar emot och behandlar. Med denna metodologi kan ägare och användare indirekt interagera med en IoT-apparat vilket är betydligt säkrare än en direkt förbindelse. För att användare skall kunna hitta ett IoT-orakel dynamiskt så är alla apparaters mellanvara också uppkopplad till samma Whisper-kanal. När mellanvaran tar emot en tillgänglighetsfråga så jämför den frågans villkor med sin egen konfiguration och svarar jakande om en likhet hittas.

Slutsats

Detta examensarbete visar att Ethereum-plattformen och blockkedjeteknologi definitivt kunde användas för att förstärka säkerheten för IoT-sektorn. Lösningen har dock nackdelar i form av en högre drivkostnad och latens. Den försedda implementationen är bara en prototyp av ett system vars primära syfte är att demonstrera hur abstrakta koncept kan praktiskt tillämpas. Flera delmoment av projektet är ännu icke-implementerade eller underforskade vilket innebär att produkten inte heller är produktionsklar. Frågor gällande verifikation och kryptering av uppdragsresultat är väldigt komplexa och skulle sannolikt kräva en sorts konsensusalgoritm att lösa.

Oavsett deras unga åldrar ser både sakernas internet och blockkedjeteknologi väldigt lovande ut för flera olika ändamål, men det är ännu omöjligt att förutspå potentiella problem i deras skalbarhet. Ethereum lyckades nyligen samla ihop den krävda mängden kryptovaluta för att sätta igång deras 2.0 version av blockkedjeprotokollet som behandlar flera framträdande problem i arkitekturen. Förändringen är lovad att göra smarta kontrakt mer användarvänliga och effektiva.

Källor

Nakamoto, S., 2008. *Bitcoin: A peer-to-peer electronic cash system*.

Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "*An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends*," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 557-564, doi: 10.1109/BigDataCongress.2017.85.

Buterin, V., 2014. *A next-generation smart contract and decentralized application platform*. white paper, 3(37).

Git-scm.com. (2019). *Git - Documentation*. [online] Tillgänglig: https://git-scm.com/doc.

Swcregistry.io. (2020). *Smart Contract Weakness Classification and Test Cases.* [online] Tillgänglig: https://swcregistry.io/ [Hämtad: 3 Dec. 2020].

Ethereum Wiki. (2020). *whisper-overview.* [online] Tillgänglig: https://eth.wiki/concepts/whisper/whisper-overview [Hämtad: 3 Dec. 2020].

Newman, P. (2020). *THE INTERNET OF THINGS 2020: Here's what over 400 IoT decision-makers say about the future of enterprise connectivity and how IoT companies can use it to grow revenue.* [online] Business Insider. Tillgänglig: https://www.businessinsider.com/internet-of-things-report?r=US&IR=T [Hämtad: 3 Dec. 2020].

Raj, E., Westerlund, M. and Espinosa-Leal, L., 2020. *Reliable Fleet Analytics for Edge IoT Solutions*. CLOUD COMPUTING 2020, p.55.