Secure IoT Applications Using Scalable Blockchain Models And PQ Primitives

Sichere IoT Anwendungen unter Nutzung skalierbarer Blockchain Modelle und PQ Primitive

Master-Thesis von Muhammad Rameez
Matriculation No.: 2556345
Tag der Einreichung:

1. Gutachten: Prof. Johannes Buchmann
2. Gutachten: Dr. Rachid El Bansarkhani
Betreuer: Dr. Rachid El Bansarkhani

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Secure IoT Applications Using Scalable Blockchain Models And PQ Primitives
Sichere IoT Anwendungen unter Nutzung skalierbarer Blockchain Modelle und PQ Primitive

Vorgelegte Master-Thesis von Muhammad Rameez
Matriculation No.: 2556345

1. Gutachten: Prof. Johannes Buchmann
2. Gutachten: Dr. Rachid El Bansarkhani
Betreuer: Dr. Rachid El Bansarkhani

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den September 20, 2018

(Muhammad Rameez)

Abstract

Distributed Ledger Technologies like blockchain has emerged as a promising area of research in academia and business. Its tamper resistant nature combined with other properties such as immutability, transparency and byzantine fault tolerance make it particularly interesting for applications in Finance, Internet of Things, Supply Chain Management, and Cloud Storage.

In this thesis, we first introduce the basics of blockchain and its related terminologies. Then, we highlight some of challenges faced by this promising new technology along with some potential solutions to those problems. After which, some choice examples of blockchain applications are presented. Next, we focus on the Ethereum blockchain, IPFS and Raiden Network and explore their potential in building new decentralized system for Supply Chain Management and Shipment Tracking.

As part of this thesis, a novel decentralized Supply Chain Management System was designed, implemented and tested. The design of this system was realized using the Ethereum blockchain and was evaluated under various scenarios designed to simulate real world application and usage. Its design has several key advantages over traditional systems. It is secure against distributed denial of service attacks (DDOS) and has additional advantages of being trustless, autonomous, transparent and censorship resistant.

Contents

1 Introduction And Motivation

1.1 Introduction To Blockchain

"Blockchain is a continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block contains typically a hash pointer as a link to a previous block, a timestamp and transaction data" [Wik18b]. It can serve as a distributed ledger that can record transactions without a central server or trusted third party. The transactions are available to all parties and are easily verifiable. It is inherently resistant to data tampering as altering data in any one block breaks the chain and requires that all subsequent blocks be calculated again using the new data. Technical details of blockchains are discussed in chapter 2, however for a high level overview please refer to figure 1. Notice that each block has a unique signature or hash and is linked to previous blocks through its hash. Blockchain has the power to revolutionize how business is conducted in digital age. Some are calling it the most important innovation since the development of the internet and the world wide web. The proponents of this technology believe that it will fundamentally transform the web itself. Internet of tomorrow will be powered by decentralized applications or Dapps [Nat18]. The first blockchain was Bitcoin, it was invented by a person or group of persons known only by the pseudonym Satoshi Nakamoto. Bitcoin is a form of peer-to-peer electronic cash designed to transfer value between two parties without involving banks or other financial institutions. It was the first to solve the double spend problem in digital currency. Bitcoin paved the way for exponential growth in crypto currency market which together with other Altcoins is worth over 120 billion dollars at the time of writing. The underlying technology which powers Bitcoin, Ethereum and other crypto currencies can be used for much more than just transferring X amount of coins from Person A to Person B. Researchers are employing blockchain technologies to increase efficiency and reduce costs in industries such as Supply Chain Management, Internet of things, Banking and Finance just to name a few.

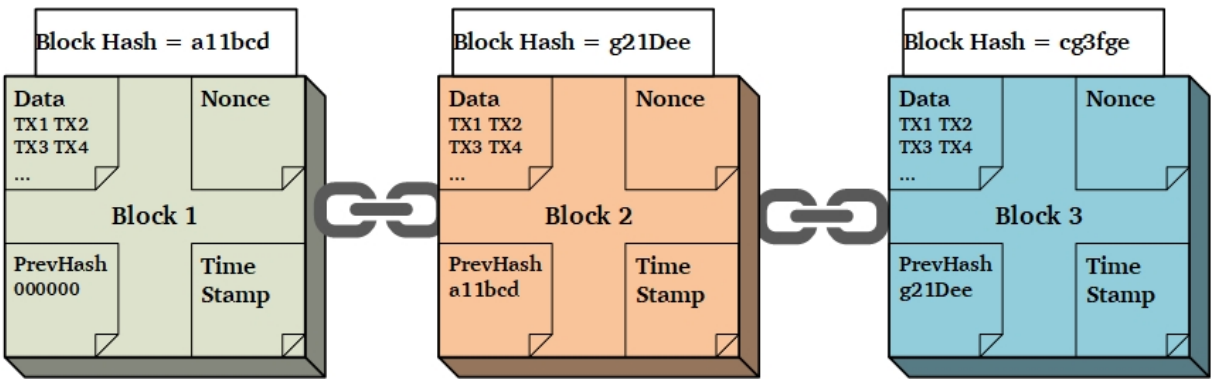

Figure 1: High Level Overview of the Blockchain

1.2 Blockchain Types

Blockchain technologies have experienced rapid growth in the recent years. The rapid pace of innovation has given rise to new models, paradigms and technologies in this sector. We now have multiple competing blockchain networks and solutions jostling for different segments of the market. There is Monero and Zcash focusing on privacy oriented solutions employing technologies like ring signatures [She14] and zk-SNARKS. Others like Ripple and Hyper Ledger are tailoring their solutions for specific segments of the market like banking and enterprise. This accelerate pace of innovation has ignited a fierce debate in the community as to what can and cannot be classified as a blockchain. The classification schemes consider range of factors including design and architecture of the underlying blockchain protocol. In the paper presented in [XWS+17] the authors classify blockchain systems based on their architecture and the level of decentralization i.e. Fully Centralized, Partially Decentralized, and Fully Decentralized. Most publications and experts however, classify blockchains into two broad categories: Public or Permissionless Blockchains, and Private or Permissioned Blockchains [Blo16a]. This classification is useful as it uses a range of functional and non-functional properties of blockchain systems for categorizing them.

	Permissionless Blockchains (Public)	**Permissioned Blockchains (Private)**
Network Access	Any one can join the network	Only authorized participants can join
Minning	Any one can produce new blocks	Authorized block producers
Decentralization	Fully decentralized system	Can have limited to no decentralization
Speed	Slower compared to private blockchains	Faster
Transparency	Fully transaparent	Customizable level of transparency

Table 1: Permissioned vs Permissionless Blockchains

1.2.1 Permissionless Blockchains

Permissionless or Public blockchains are highly decentralized networks that value decentralization and censorship resistance above everything else. Permissionless systems allow any person or entity to interact with the network or run smart contracts [Kar18]. Every participant has equal rights to create, send, and view transactions. They can even choose to become block producers or miners and verify transactions in order to append them to the blockchain.

Bitcoin was the first public blockchain and a realization of a vision outlined by Satoshi in his white paper [Nak08]. He identified decentralization, censorship resistance and distributed trust as the most important factors for the success of the peer-to-peer digital cash protocol he proposed. The main functional and non-functional properties of permissionless systems are given below:

Decentralization: Public blockchains are usually highly distributed and decentralized. This makes the network censorship resistant and harder to attack and bring down by any one entity.

Consensus Mechanisms: Decentralization is achieved at the cost of higher complexity hence public blockchains requires sophisticate consensus mechanisms. There are two main types of consensus mechanisms: Proof of Work [2.6.1], and Proof of Stake [2.6.2]. The process of reaching consensus and extending the blockchain is known as mining. Any participating node can run the mining software in order to verify transactions and extend the blockchain [Kar18], [Blo16a].

Block Producers: Any node can choose to become a block producer or miner. Public blockchains usually employ a crypto economic model where by miners are rewarded using network assets or tokens [Nak08].

Privacy: Without a central entity or coordinator transparency becomes an important feature for the participants and miners in order for them to trust the system. This transparency is often achieved by sacrificing some degree of privacy. By default, all transactions and data in a blockchain is public and can be easily accessed with the help of any block explorer.

1.2.2 Permissioned Blockchains

Permissioned or Private blockchains are sometimes also referred to as private blockchains. It is a closed echo system where participants need permission from an administrator or special node to interact with the ledger. Only pre-approved nodes or block producers can verify transactions and run smart contracts [Kar18]. Participants place some level of trust in these block validators or administrators.

Permissioned blockchains can be run by members of a consortium in order to increase transparency and efficiency of inter organizational processes. They allow organizations to have better control over proprietary data while facilitating trusted exchange of secure information across organizational hierarchy [Kar18], [Blo16a]. The main functional and non-functional properties of permissioned blockchain systems are given below:

Decentralization: Permissioned systems can have varying degrees of decentralization. They can be fully centralized or partially decentralized. Systems like Hyperledger fabric allow fine grained control over governance models and the level of decentralization. In the end the level of decentralization in permissioned systems depends upon many factors including number of participants, their relationship with each other, degree of required fault tolerance, business rules and consensus algorithms participants agree on [Kar18], [Blo16a].

Consensus Mechanisms: Permissioned blockchains usually do not need to run complex consensus algorithms. They can run simplified consensus mechanisms. This coupled with the fact that only a limited number of nodes are responsible for producing new blocks helps them become more efficient and scalable [Kar18], [Blo16a].

Block Producers: Private blockchains can set out criteria for participants to become block producers. These criteria can be based on certain business rules or participating nodes might be required to meet special conditions in order to become block producers such as demonstrating certain capabilities like minimum hash power, or having possession of certain assets etc. Unlike public blockchains block producers are not rewarded with network assets rather they work together to increase efficiency, reduce business costs and boost productivity [Kar18], [Blo16a].

Privacy: They can offer fine grained control over transaction visibility as opposed to public blockchains where basically any one can view any transaction by simply querying the blockchain or using a block explorer. This is a huge incentive for organizations to use permissioned blockchains as they might wish to prevent unauthorized disclosure of sensitive information [Kar18], [Blo16a].

1.3 Motivation

Blockchain has exploded as the technology of the future for several industries including cross border payments, peer to peer transactions, regulatory compliance, healthcare and supply chain management [Ber18]. It provides a tamper proof immutable ledger which can be particularly useful in tracking goods and services as they move and change hands across borders in the supply chain. It enables new and innovative means of organizing and tracking data. In the modern era industries are highly interconnected through complex supply chains with their partners and suppliers across the globe. The success of a supply chain depends upon the integration and coordination of all of its participants. Consider the example of an Airbus A380 which is made up of four million individual parts and is built in six different sections in plants around Europe. Its wings are manufactured in Wales, the fuselage comes from Hamburg, Germany and the final assembly takes place in Toulouse, see figure 2. This cross border and federated model of manufacturing is possible only through just-in-time manufacturing and supply chains. Airbus and other multinational companies depend on JIT to ensure that their products and services are competitive in the global market. JIT processes depend upon sophisticated supply chain management and Inventory tracking systems to maximize cost-efficiency and minimize delays. Transparency, efficient communication and quick dispute resolution are key to the success of any supply chain. Traditional supply chain management systems are mostly centralized and siloed inside organizational structures. These systems are highly dependent on human actors to update the state of the system. This can lead to side effects such as increased complexity, reduced efficiency, and human errors.

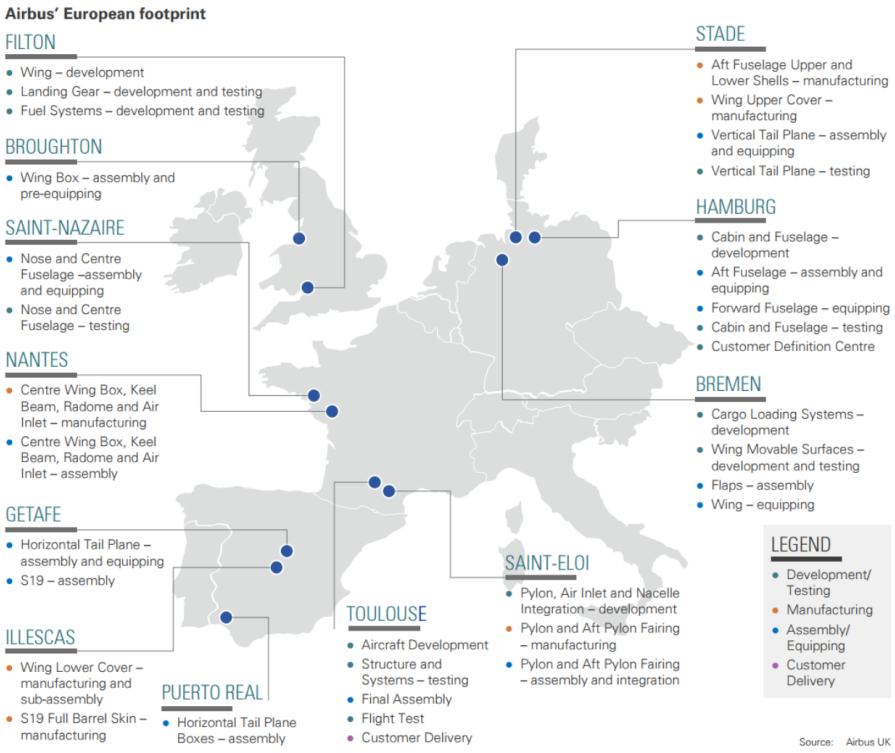

Figure 2: Integrated Supply Chains [Air18]

Blockchains can offer numerous benefits for streamlining processes and increasing transparency across the supply chain. Coupled with Smart Contracts and IoT, supply chain can become one of the killer applications of the blockchain. Smart Contract platforms such as the Ethereum can use tracking data to automate various functions and events in the supply chain life cycle. Ethereum's distributed ledger also provides total transparency to all parties involved. By increasing automation within supply chain processes they can reduce complexity and eliminate errors and delays. Some of the key benefit of blockchains in the context of SCM are follows:

Transparency: Its shared ledger enables all stakeholders to have the same view of the data stored on the blockchain. Transparency is greatly increased due to everyone having real time access to the same data.

Traceability and Compliance: Every transaction recorded on the blockchain is cryptographically verified. This increases traceability, reduces chances of fraud and counterfeit products, and helps to increases compliance for exisiting products.

Security: Any system built on the blockchain is by design highly secure against DDoS and single points of failure. Each transaction on the blockchain is replicated across multiple nodes on a distributed ledger. Each block links to a previous block hence any attacker wanting to modify data in any block will need to modify all subsequent blocks as well.

Trust: Most traditional SCM systems allow participants a very limited view throughout the supply chain life cycle. Usually participants only have access to information necessary to successfully realize the next process of the supply chain. This creates information asymmetry between different stake holders. Decentralized blockchain based SCM systems can allow participants to have same view of the entire system and hence reduce information asymmetry and increase trust.

1.4 Thesis Objective

The primary goal of this thesis is to design and develop a secure and decentralized supply chain management and tracking system. The state of the art for this system will be a smart contract for monitoring supply chain cycle under strict conditions and a decentralized application designed to interact with the smart contract in order to automate supply chain processes. In order to realize this system following questions and issues must be addressed:

(a) Which blockchain platform is best suited for development of this system?

(b) How to reduce complexity and increase automation in supply chain processes?

(c) Is it possible to improve or enhance the blockchain security model by using post quantum primitives?

The design and requirements of this system are based on a well quantified use case scenario presented in chapter 5.

2 Blockchain: Technical Primer

In this chapter some of the technical concepts and terminologies related to blockchain are explained. This technology allows participants to transact with each other using a peer-to-peer network that guarantees censorship resistance, immutability, transaction finality, and protection against double spend attacks. In order to better explain how it works consider figure 3. In this example Alice wants to transfer two bitcoins to Bob. She uses her private key to creates a signed transaction for transferring these coins to Bob. Every user in the network has pair of keys; a public key that serves as their unique identifier or address and a private key for signing transactions (see section 2.2). The signed transaction is broadcast to the bitcoin network where it waits until it is picked up by a special node called a miner. The miner verifies transaction signatures and batches pending transactions into blocks. Each block carries the hash of the one that came immediately before it. Next step is to calculate the hash of the entire block and append it at the end of the blockchain as shown in figure 3. Blockchain protocols have built in consensus mechanism to ensure that peers always agree on only one longest chain section [2.6]. Blockchain is not just a technology it is actually a systems and like most systems it is composed of individual components which come together to make the whole. The next few sections explain the important building blocks or sub components of the blockchain system.

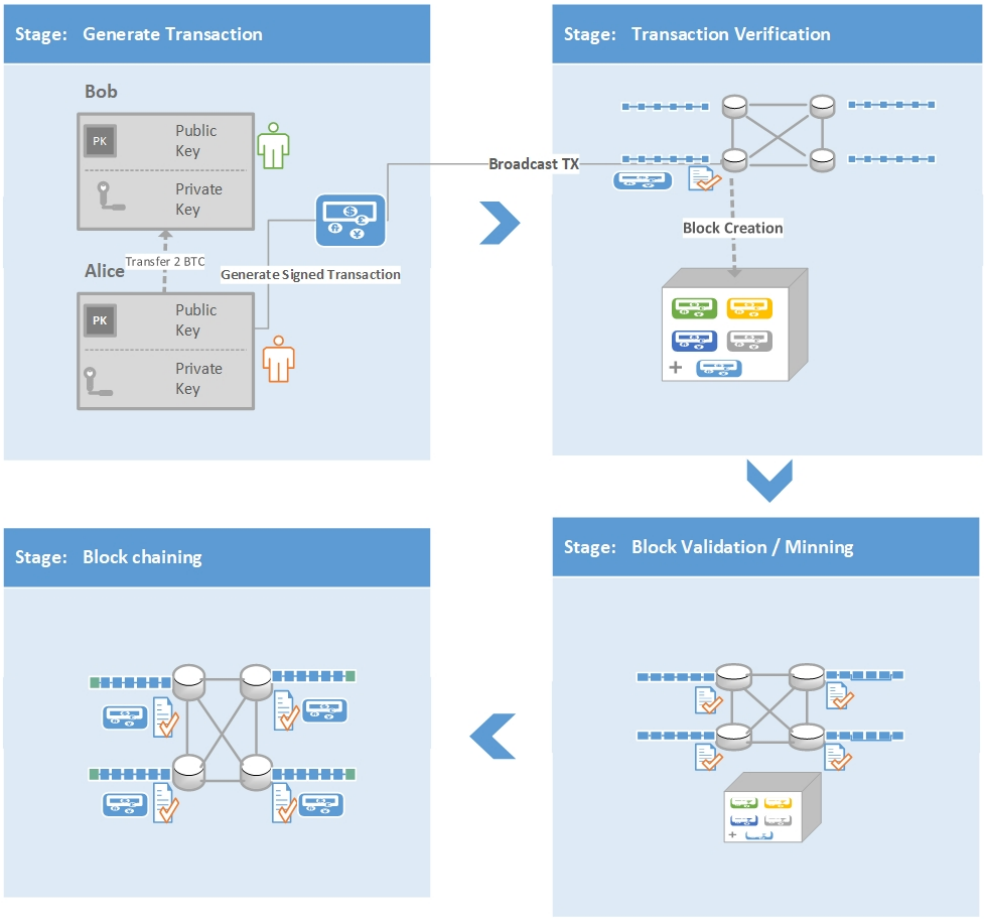

Figure 3: Transaction life cycle in a Blockchain

2.1 Distributed Ledger Technology

Distributed ledger Technology refers to a shared and distributed database replicated across members of a peer-to-peer network. Each member of the network receives the same copy of the data. New data can only be added to the ledger when consensus is achieved among the members. Consensus rules and mechanisms [2.6] may vary from network to network. These rules are designed to ensure that data on the ledger remains synchronized across network participants. Blockchain is a special type of distributed ledger where cryptography is used to achieve consensus and ensure transaction authenticity. Information stored on the blockchain is immutable i.e. once recorded it cannot be altered. Blockchain is not the only structure used for DLT. IOTA [Wik18a] and Hashgraph have successfully employed Directed Acyclic Graphs (DAG) [Wik18c] for creating their DLT.

2.2 Asymmetric Cryptography

Asymmetric or public key cryptography is an important building block of any blockchain network. This cryptography technique relies on a pair of keys: A public key which is widely available or shared with everyone, and a private key which is only known to the owner. These two keys are mathematically related to each other, in that one key usually encrypts and the other key is used for decryption. If the private key encrypts only the corresponding public key can decrypt and vice versa [Wik18e]. Public key cryptography is widely employed for:

Public key Encryption: is an encryption technique where data is encrypted using senders public key. The recipient can only decrypt the data and read the message if he is in possession of the corresponding private key [Wik18e]. Encryption guarantees confidentiality. Encryption function $Encrypt(m, pk) \rightarrow C$ takes an arbitrary length plain text message and converts it to cipher text using the recipients public key. The recipients applies the decryption function $Decrypt(C, Sk) \rightarrow PlainText$ to decrypt the encrypted message using his secret key.

Digital Signatures: based on public key cryptography are used in a number of applications including blockchain. Data or message is signed with sender's private key. Anyone can verify the message signature using the corresponding public key. In Bitcoin, Ethereum and other blockchain networks digital signatures are used to guarantee authenticity, integrity and non-repudiation [Wik18e].

2.3 Cryptographic Hash Functions

Hash functions underpin most cryptographic primitives of modern security applications. Hash Functions are used to calculate fixed length hashes for variable length data. Formally a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ is computed as $\mathscr{H} = (H, KGen)$, where $KGen(1^n)$ outputs a fixed length byte string $H(m)\ \epsilon\ \{0,1\}^n$ given a variable length message m, such that $m\ \epsilon\ \{0,1\}^*$ [BM18]. Hash functions

are used for message authentication (HMACS), and digital signatures and other forms of authentication. Hashing algorithms are designed in a way that even a slight change in the input data will result in a vastly different output hash [RS04]. All cryptographic hash function should have the following basic properties:

Deterministic: The hash function should be deterministic. This means that for any given input m that produces a hash value $H(m)$, the hash function should always produce the hash value $H(m)$ as long the input remains the same [RS04].

One-Wayeness: This property implies that it is not realistic or computationally feasible to determine the input message m given a random $H(m)$ [BM18]. In other words the hash function is a One-Way function [RS04].

Collision Resistance: A hash functions needs to be collision resistant, if it is unfeasible to find two messages, such that $m \neq m'$ that map to the same hash value $H(m) = H(m')$ [BM18]. In other words it is very difficult to find two different inputs which hash to the same output hash value.

2.3.1 Hashing and Blockchains

Blockchain networks employ hashing algorithms in their consensus protocols and mining (see 2.6) process. Ethereum uses an algorithm called Ethash (modfied Dagger Hashimoto) and bitcoin uses SHA-256 [Quy15] hashing algorithm. These algorithms are used as a mechanism to guarantee integrity and to prevent unauthorized tampering and corruption of the distributed ledger. They are used to link blocks with each other in the blockchain. Each new block contains the hash of the blockchain that came before it as shown in figure 1. Block hash represents the state of the blockchain when it was created. This allows anyone to easily verify the complete state of the entire blockchain. Any attempt to alter data in any block will result in a vastly different hash for that block and will also require that hashes for all subsequent blocks be recalculated.

2.4 Digital Signatures

Digital Signatures are an important building block of most cryptographic protocol and security systems. They are used for guaranteeing data integrity, authentication, and non – repudiation. Data Integrity means that the receiver can have confidence that the data was not altered during transit. If the message has a signature attached the receiver can calculate the signature on the message to see if it matches the signature attached with the message. Authentication guarantees that the message came from authenticated source i.e. sender of the message is in fact the one holding the private key that signed this message [RSA78]. Non – repudiation means that the sender cannot deny that he sent the message if it is correctly signed. A digital signature scheme includes a digital key generation algorithm, a secure

signing algorithm and a secure verification algorithm. The key generation algorithm G generates public/private key pairs. It is given by $G : keyGen(1^n) \rightarrow (Pk, Sk)$. The secure signing algorithm S is given by $S : Sign(sk, m) \rightarrow \sigma$, and the verification algorithm accepts or rejects the signatures, it is given by $V : Verify(pk, m, \sigma) = \{0, 1\}$ [Lys02]. For correctness S and V must satisfy:

$$Pr[(pk, sk) \leftarrow G(1^n), V(pk, m, S(sk, m)) = accepted] = 1 \tag{1}$$

Digital Signatures were first introduced by Rivest, Shamir and Adleman in their paper [RSA78] as part of the RSA digital signature scheme. This algorithm uses public key cryptography to sign message digest [RSA78].

2.4.1 Digital Signatures and Wallets

Digital Signatures are used to verify the authenticity of a transaction in blockchains. A transaction is generated using a program known as a digital wallet. Digital wallets manage user keys and helps them transfer coins. Transactions are signed using some signature algorithm. Bitcoin and Ethereum use Elliptic Curve Digital signature algorithm for signing transactions. The wallet regularly queries the blockchain to get number of coins assigned to a user's private key. Figure 4 shows how digital signatures are used in the blockchain. Alice generates a signed transaction for transferring 1 btc to Bob. This transaction is broadcast to the network where it is picked up by a miner. The miner verifies Alice's signature using her public key. If the signature is verified the miner updates the blockchain ledger with new balances for Alice and Bob.

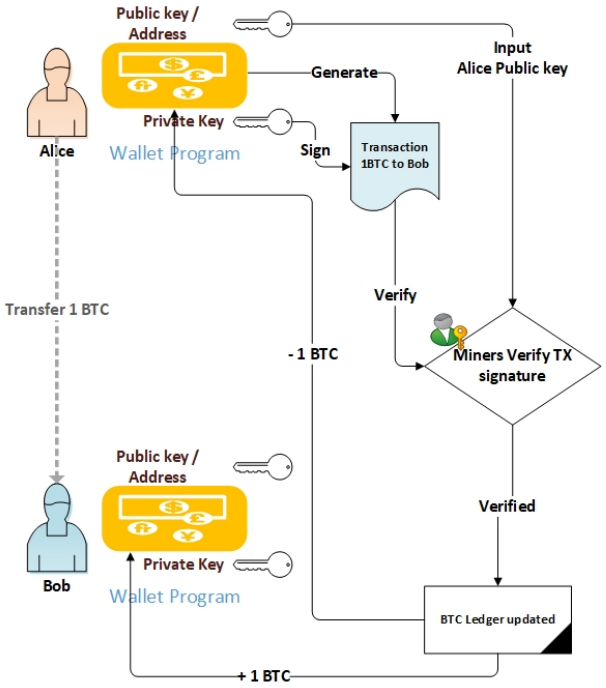

Figure 4: Digital Signatures in Blockchain

2.5 Merkle Trees

Blockchain is a continuously growing list of transactions which are grouped into blocks. A block contains multiple transactions and a block hash as shown in figure 1. This hash is calculated over the entire block. A block is actually a special data structure which is implemented with the help of a Merkle tree as shown in figure 5. Andreas Antonopoulos defines merkle trees in his book "Mastering Bitcoin" as *"A merkle tree, also known as a binary hash tree, is a data structure used for efficiently summarizing and verifying the integrity of large sets of data."* [And14] (ch.9). They are used to summarize all transactions in a block using cryptographic hashes. Transactions are grouped into blocks and hashed together to form a Merkle tree [Nak08]. The root of this Merkle tree or Root hash is stored in the block's header This process is shown in figure 5. This enables fast and efficient verification of any transaction in a particular block. In a tree comprising of N data elements only $2 * log2(N)$ calculations are required to verify if a particular data element or transaction is included or not. Without Merkle trees it will be prohibitively expensive to run blockchain nodes, which would severely impact the decentralization of the system [BM18], [Nak08].

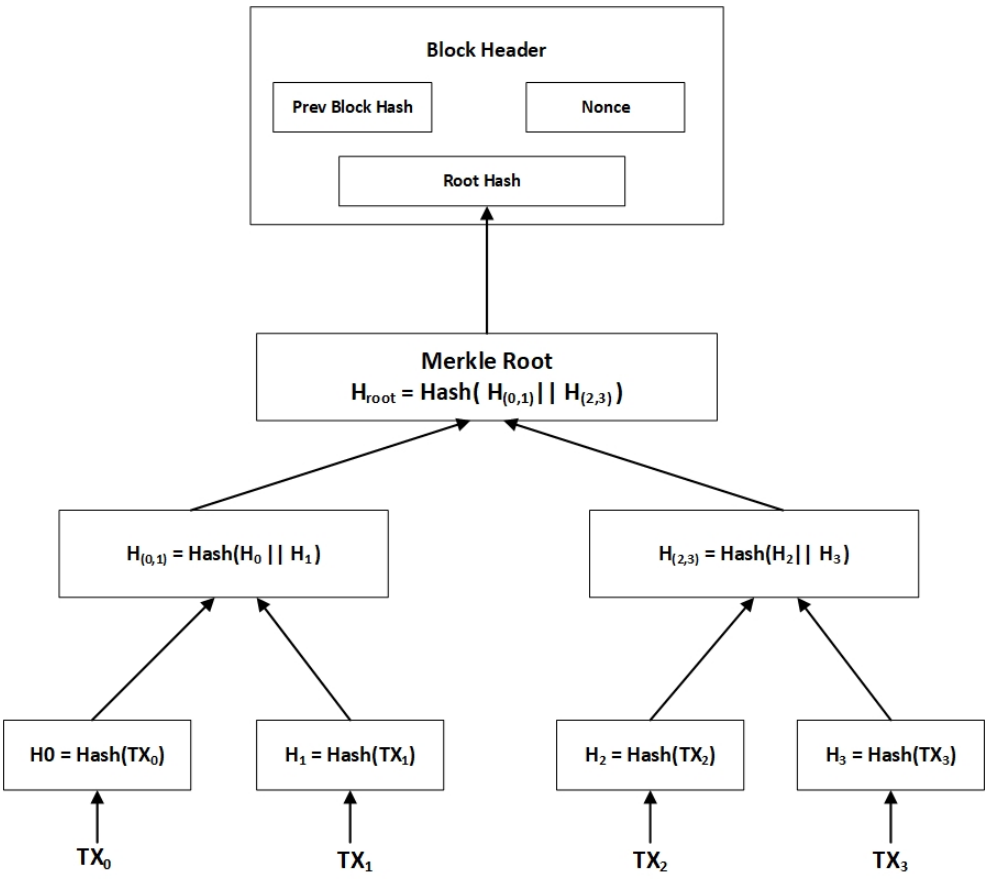

Figure 5: Merkle Root Summarizes Transactions in a Block

In bitcoin a Merkle tree is constructed by recursively hashing pair of nodes using SHA256 cryptographic hash function as shown in figure 5 [And14] (ch.9). In the example tree there are four leaf nodes storing

hashes of four transactions. The leaf nodes do not store actual transactions rather TX data is hashed and result is stored in the Merkle tree. Each leaf node is designated as H_0, H_1, H_2, H_3 and given by the equation:

$$H_0 = SHA256(SHA256(Transaction_0))$$

Since Merkle trees are in essence binary trees hence even number of nodes are required to have a balanced tree. Two leaf nodes are hashed together to form a parent node as given by equation (2). In the event of odd number of transactions, the last transaction is duplicated to have an even number of leaf nodes equation (3). The recursive hashing process starts from the bottom and continues until there is only one node left at the top which is called the Merkle Root. This is the parent hash of all child nodes and summarizes all the data in all transactions. The root hash is placed in the block header [BM18], [Nak08].

$$H_{(0,1)} = SHA256(SHA256(H_0||H_1)) \tag{2}$$

$$H_{(3,3)} = SHA256(SHA256(H_3||H_3)) \tag{3}$$

2.6 Consensus Mechanisms and Protocols

Electronic coins are defined as a chain of digital signatures that serves to establish ownership [Nak08]. In order for Alice to transfer one coin to Bob, she must sign the hash of a previous transaction and the public key of Bob. Anyone can verify the chain of ownership by verifying signatures. Signature verification only confirms that Alice was in possession of the coin at some point in time. It does not guarantee that Alice did not try to spend the same coin more than once [Nak08]. Therefor a mechanism is needed that guarantees that any previous owner (Alice) did not sign any earlier transaction for transferring the same coin. The only way to verify this in a decentralize system is to announce all transactions, and to have a mechanism to ensure that all participants agree on a single shared history or order of transactions. Payee (Bob) needs proof, that at the time of token transfer, the majority of network participants agreed, that Alice did not generate an earlier transaction to transfer the same coin [Nak08]. The process which establishes consensus among network participants is called Mining.

Mining

Blockchain participants must agree on a single state of the distributed ledger. In Bitcoin the process of achieving consensus is termed as Mining. Mining underpins bitcoin's security model. This process is a designed to guarantee security and integrity of the distributed ledger. It serves to protect the network

from fraudulent transactions and double spend attacks [Nak08]. It is also the process by which new blocks are generated. Miners spend something of value like electricity in the form of computing power by running complicated algorithms (Proof of Work 2.6.1). They are rewarded by the network with block rewards or newly minted coins. This process prevents bad actors or attackers from modifying the state of decentralized ledger against network rules [Nak08]. Attackers will need to control at least fifty-one percent of network hash rate to mount a successful attack. This is virtually impossible in a sufficiently large decentralized network like Bitcoin or Ethereum. There are two main type of consensus algorithms Proof-of-Work 2.6.1 and Proof-of-Stake 2.6.2.

2.6.1 Proof-of-Work

It was proposed by Satoshi Nakamoto in the bitcoin white paper [Nak08], as means for establishing consensus. Miners solve complicated mathematical problems to validate transactions. Pending transactions are batched into blocks and the miners compete with each other to calculate the hash of the block. The hash output of the block must start with specific number of leading zeroes in order to satisfy protocol rules. The exact number of leading zeroes depend upon the network difficulty which is adjusted automatically after every 2016 blocks. This difficulty determines how easy or hard it is to find the output hash for a block. The function that calculates difficulty is determined by a moving average and targets average number of blocks per hour. If blocks are generated too fast, the difficulty increases and vice versa. This is done in order to compensate for increasing hardware speed and varying interests in running nodes. Proof-of-work protocols can be summarized in the following steps [Jim18]:

- Miners try to find the hash output for a block with a fixed number of leading zeroes. They do this by repeatedly changing part of the block called nonce and recalculating the hash output.

- First miner to solve the puzzle and find the hash broadcasts his solution or proof-of-work to the rest of the network.

- Upon receiving the solution other miners verify it to ensure that it is correct. Before they agree to add the it to the blockchain they verify all the transaction in the block to make sure they are valid.

- If majority of miners agree on the solution and agree to add the block to the blockchain than consensus is achieved.

This approach has some inherent disadvantages. It requires huge amounts of electricity to achieve consensus in large blockchain networks such as Bitcoin and Ethereum. Some estimates have put bitcoins annual energy consumption on the same level as countries likes Austria or Switzerland. POW operates on the basis of one CPU one vote model. This approach can lead to mining centralization by large mining pools and chip manufacturers.

2.6.2 Proof-of-Stake

Proof of Stake is an alternative approach for reaching consensus and protecting from double spend attacks in a decentralized network. It solves many problems inherent to POW algorithms. It is defined as *"Proof of Stake (PoS) is a category of consensus algorithms for public blockchains that depend on a validator's economic stake in the network"* [Eth18]. POS requires users or forgers as they are called to lock up their digital coins in an escrow to get a chance to validate new blocks. The deposited coins serve as collateral and an incentive for the forgers to behave honestly. If a forger approves fraudulent transaction they will lose the coins they staked and will be banned from participating in the block validation process in the future. The crux of POS systems is the fact that for any attack to be successful the attacker will need to own majority of the coins on the network. Therefore, the attacker will be the one most severely impacted by his own attack [Bit12]. This serves as a huge deterrent against any potential bad actors. Block validators are incentivized with block rewards (combination of Tx fees and coins). They are selected by the network in a pseudo – random selection process based on a combination of factors. Selecting forgers solely on the size of their stake will hugely benefit the rich miners making the rich even more richer. There are several methods to avoid these problems two of which are given below. [Max18], [Sha17]

Coin Age based Selection: This method choses validators based on how long their coins have been staked for or the 'coin age' of their stake. The coin age is calculated by multiplying the size of a validators stake with the number of days the coins have been held in escrow. Once a validator generates a block their coin age is reset and they have to wait a fixed amount of time before they can be selected to validate another block [Sha17].

Randomized Block Selection: This method choses validators based on a combination of lowest hash value and the size of their stake [Sha17].

2.7 Scaling Debate and Solutions

Blockchain technology is still in its infancy. It is a novel idea to solve several interesting problems in a trustless and decentralized manner. However, it needs to be able to scale to handle millions of transactions per second in order to compete with existing centralized platforms. Bitcoin can on average perform 3 to 4 transactions per second while Ethereum can handle up to 20 transactions per second. Compare this to Visa which on average handled over 1100 transactions per second in 2016 and has an estimated capacity to perform up to 100000 transactions in a second [Fre17]. Transaction speed measured in TPS is an important to metric to measure the performance of any financial system. During 2016 and 2017 major blockchain networks saw enormous growth in their user base. This caused exponential increase in transaction volume resulting in congested networks which caused huge delays in transaction confirmations as shown by graph figure 6. This had a domino effect on transaction fees as well, causing them to sky rocket. Miners are incentivized to pick transactions with higher fees to mine first. The long

confirmation delays coupled with high transactions fees caused many organizations and vendors to stop accepting bitcoins [Jam18]. The scaling problem is further amplified in smart contract platforms like Ethereum[4.1] which aim to be a hub for large scale decentralized applications or Dapps. Most existing and proposed Dapps use mircotransaction (MTX) as part of their business model. *"Microtransactions are a business model where users can purchase virtual goods via micropayments"* [Wik18d]. In order to efficiently run such applications Ethereum needs a way to effectively handle μ-transactions. In most cases these transactions need to be executed immediately and cannot wait for long block confirmation times. This requires exponential increase in transactions per second or TPS, for blockchain to become a viable alternative to centralized solutions. Sections 2.7.1 to 2.7.5 present possible solutions for solving the scaling problem.

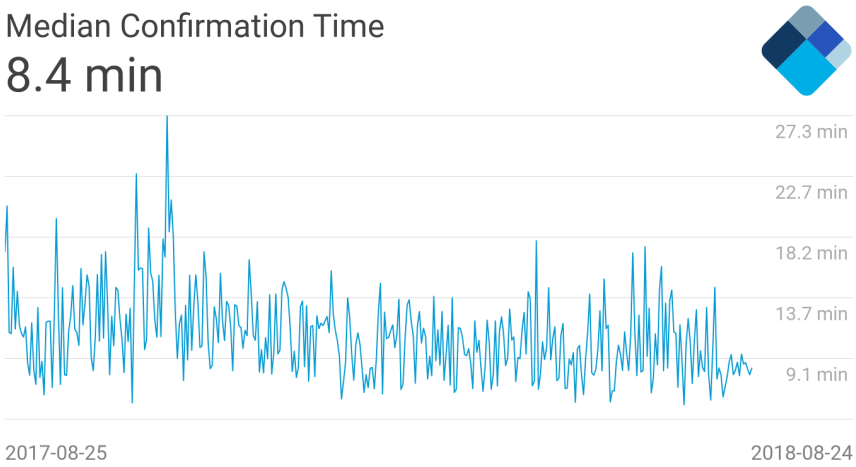

Figure 6: Median confirmation times for BTC transactions [Blo18]

2.7.1 Block Size Increase

Transactions are grouped into blocks before they are verified. POW consensus rules require that there is some distance between successive blocks so that each verified block is successively incorporated by a majority of nodes in their copy of the ledger. This means that roughly only one block is generated every ten minutes [Nak08]. In addition, some blockchains like Bitcoin have placed an upper limit on the size of each block. Currently for Bitcoin the block size limit is 1 megabyte. One suggestion is to simply increase the block limit to allow more transactions to be verified at any given time. This solution however has many problems. First of all, increasing block size results in only a linear increase in transactions per second. Secondly, it will adversely impact the decentralization of the network. Larger blocks require higher computational power to process each block and also drastically impact the size of the distributed ledger. This leads to more centralization as not everyone can afford the equipment required to successfully mine new blocks [GoC17].

2.7.2 Payment Channels - Lightning Network

An alternative solution is to use Layer - 2 transaction networks. Lightning network is bitcoins proposed solution for the scaling problem. Lightning network is defined as *"A decentralized system for instant, high-volume micropayments that removes the risk of delegating custody of funds to trusted third parties"* [JP16]. It advocates using payment channels to handle transactions off chain. Payment Channels allow users to transfer coins and assets using off chain transactions which can be enforced on the blockchain using a cryptographic technique known as hash locked transfers. Hash locking prevents spending of outputs until some specified condition is met or some secret is revealed. To guarantee security assets in Payment Channels must be backed up by assets on the blockchain. Multiple channels are joined together to form payment networks. In an ideal situation where all participants are honest only two transactions are added to the blockchain while participants can make unlimited number of instant transactions within the channel. Lightning network and Raiden network 4.3 are both examples of payment networks. Payment channels are essentially multi-signature blockchain addresses. In order to spend funds from the channel both parties must sign off on the transaction agreeing to the new balance of the channel. The new balance is stored as the most recent transaction in the channel. Simply put, a lightning network payment channel is a smart contract on the bitcoin blockchain which is mostly executed off-chain after creation. In the ideal case, the two transactions that go on the blockchain are the ones for opening and closing a channel [Lig15]. This enables users to make off chain payments with confidence. If anything goes wrong blockchain can cryptographically verify the terms of the smart contract and enforce them on-chain [Jef18], [JP16].

2.7.3 State channels – PERUN

State channels work in the same manner as payment channels in the lightning network. They can deal with any type of state altering transaction. State Channels significantly enrich the functionality of payment channels. Consider a smart contract 'C' which is executed in a state channel in an off-chain manner. This is achieved by letting channel state contain storage string α as well as financial balance. Where α describes the current state of 'C' by storing values of all contract variables. As long as there is no conflict all parties can freely update α. In the event that one of the users misbehaves the others can push the latest version of α on the blockchain which will finish executing 'C' starting from the latest agreed upon state i.e. α. A brief summary of this process is given below: [Jef18]

1. Part of the state of blockchain is locked in a multi signature address, all participants must agree in order to update this state [Jef18].

2. Instead of submitting updates to the blockchain participants update the state of the channel between themselves [Jef18].

3. Each new fully signed update overwrites the previous one and is the only valid state for the channel that can be pushed on the blockchain [Jef18].

4. In the event of a disagreement, Any participant can push the latest fully signed state to the blockchain which will finish executing the remainder of the contract and settle any outstanding operations [Jef18].

5. If nothing goes wrong participants submit the final state to the blockchain which closes the channel [Jef18].

Perun offers a new technique for connecting state channels which does not require interaction with intermediaries for every single update of the channel. Perun constructs a new primitive called virtual channels over so called multistate channels. Multi state channels are an extension of state channels presented above. Multistate channels allow for parallel creation and execution of several Nano contracts. Perun is an interesting proposal for scaling Ethereum blockchain which can potentially work for all types of smart contracts. The scheme presented in Perun claims to be secure against arbitrary corruption of any of the communicating partners or intermediaries. Detailed technical descriptions of this scheme are presented in [DFH18] and [DEFM17]. In this scheme basic state channels are connected via "virtual channels" which minimize interaction with intermediaries in channel chains. Consider an example where Alice and Bob establish a channel with each other with the help of intermediary Ingrid. In the case when they are both honest each update to the channel can be made independent of Ingrid and the only interaction that involves Ingrid is for opening and closing of the channel. In the event that a dispute arises between Alice and Bob, they first try to resolve the dispute with the help of Ingrid. If that fails, then the dispute resolution is escalated to the block chain [DFH18], [DEFM17].

2.7.4 Sharding

One scaling proposal that is of particular interest, specially for the Ethereum community is Sharing. Sharding is not a new concept in fact it has been successfully employed in large database systems for a number of years now. A database shard is a horizontal partitioning of the database. Each shard is held on a separate server. This spreads the load and improves performance [CDE+12]. Ethereum developers are proposing to apply the same concept to blockchain space. Their proposal calls for partitioning the Ethereum Blockchain into smaller chunks or shards. The concept is to partition Ethereum nodes and transactions into smaller groups. Each group is responsible for and manages a section of the ledger instead of every node being responsible for the complete ledger. A smart contract will be used to manage data and transactions that are accepted as valid by the main chain. The proposal further calls for creating a set of notaries to vote on validity of transactions within each shard [Eth17a].

2.7.5 Sidechains

Sidechain is a blockchain that runs parallel to the main blockchain. It extends the functionality of the main chain enabling decentralize transfer of assets and tokens between the two chains. Sidechains allow coins to be moved between two separate blockchains. Tokens from the main chain can be securely moved to the sidechain and used in these chains. The token transfer takes places at a fixed predetermined rate [RIC14]. In order to transfer main chain assets to a side chain they are sent to a special address on the main chain. Once this transaction is verified a confirmation is broadcast in the sidechain enabling the network to assign equivalent assets to the users account in the sidechain [Ada14]. They can help with blockchain scaling as they can take some of the pressure off the main chain. Developers can design specialized sidechains to run their Dapps more efficiently while still taking advantage of security and decentralization provided by the mainchain. Sidechains are implemented using a Two – way peg as shown in figure 7.

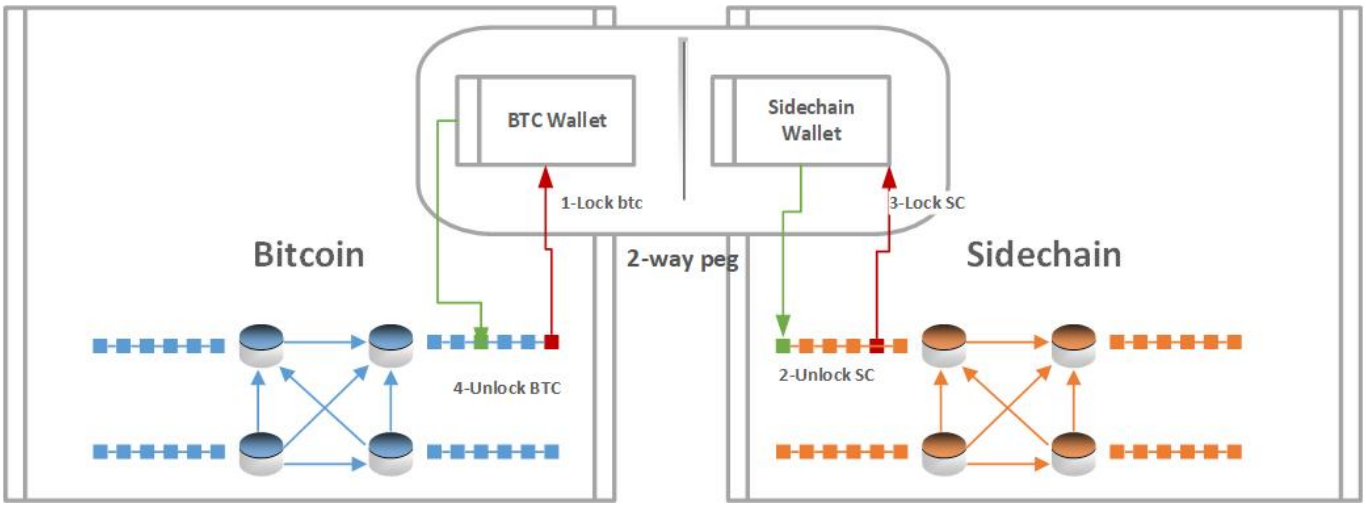

Figure 7: Two way Pegged Sidechain

3 Blockchain Applications

Companies and People across the globe are exploring applications of blockchain technology across several industries. This chapter details blockchain applications and projects from various sectors including finance, Internet of Things, Supply chain Management and File hosting and sharing.

3.1 Crypto Currencies

Bitcoin was the first ever application of blockchain technology. It was proposed and developed shortly after the financial crash of 2007. This crash was caused by the irregularities in the existing centralized financial institutions and banks. The biggest problem with traditional banking is that it centralizes trust in a few large financial institutions and banks. This system works only as long as these banks operate reliably and responsibly. The financial crash of 2007 showed that they cannot be always trusted to act responsibly [Pat12]. Bitcoin was developed to address problems prevalent in existing centralized financial system. It solves these problems by developing a system which removes all central entities and trust is established through a system of checks and balances [2]. Cryptocurrency sector experienced huge growth in terms of users and investment during 2016 and 2017 and at one point was worth close to 500 billion dollars [Wil12]. Unfortunately, most of this growth was due to speculative investments and has not yet translated into large scale adoption of Cryptocurrencies in everyday business and finance. Blockchain powered financial systems have huge promise provided they are able to solve some of the challenges outlined in section 2.7.

3.2 Internet of Things

IoT is the next wave of automation promising to disrupt industrial and domestic structures and processes. The billions of smart devices coming online could transform homes, cities, offices and factory floors [IBM16]. *"IoT holds the promise to expand business processes and to accelerate growth. However, the rapid evolution of the IoT market has caused an explosion in the number and variety of IoT solutions, which created real challenges as the industry evolves, mainly, the urgent need for a secure IoT model to perform common tasks such as sensing, processing, storage, and communicating"* [Ahm16]. Currently IoT ecosystems are realized using brokered communication models based on client/server paradigm. Devices are connected through cloud servers using the internet even if they are few feet away from each other. Further more centralized models struggle to scale up to meet the demands of billions of users or devices [Ben16]. Blockchains offer an intriguing alternative as a secure and decentralized IoT command, control and communication model. Blockchain and Smart contract based solutions should be more manageable and scalable then traditional ones. Blockchains are inherently tamper resistant hence they will prevent one or more rogue devices from causing a complete system breakdown across a home, factory, or transportation system. Blockchain in IoT sphere has the potential to revolutionize several industries and businesses [IBM16], [Ben16], [Ahm16]. There are several exciting projects working

towards this goal. Sections 3.2.1 and 3.2.2 describe two projects at the forefront of blockchain and IoT.

3.2.1 Autonomous Decentralized Peer-to-Peer Telemetry (ADEPT)

ADEPT is a joint venture between IBM and Samsung Electronics. It is developed to serve as a validation platform for projects proposing to connect IoT and Blockchain. It envisions network of devices that are capable of autonomously maintaining themselves. Adept is working towards integrating IBMs Watson IoT platform with blockchain technologies. This project is still in development stage but proof of concept has already been implemented [Pur16]. It uses blockchains as the backbone of the system using a mix of proof of work and proof stake to secure transactions. The ADEPT architecture supports three foundational functions.

- Peer to Peer encrypted messaging using a secure messaging protocol called TELEHASh [Pur16].
- Decentralized file sharing based on BitTorrent protocol [Pur16].
- Decentralize device coordination and control In the absence of centralize controller, device control and coordination becomes significantly challenging. Adept uses Ethereum or hyper ledger blockchain to implement this in a trustless secure fashion [Pur16].

Ultimately it will enable IoT devices to send data to private blockchain ledgers for inclusion in shared transactions with tamper-resistant records. Devices will be able to communicate with the blockchain in order to update or validate smart contracts. This will improve transparency and reduce conflicts by empowering all stake holders. Each stakeholder will have access to the same data and could easily verify all transactions [Pur16], [Ben16].

3.2.2 Filament

Filament is a technology stack that *"enables devices to discover, communicate, and interact with each other in a fully autonomous and distributed manner"* [Fil16a]. Its main focus is industrial Internet of Things. *"The Filament technology stack is built upon five key principles: Security, Privacy, Autonomy, Decentralization, and Exchange (SPADE)"* [Fil16a]. Filament uses Telehash for secure encrypted device to device communication. Secure Identity is provided by blockchain. Once a secure communication channel has been established between devices, smart contracts are used to interact with them, or to enable them to transact with each other. Smart Contracts in Filament run directly on device and accept or run transactions from other devices based on contractual terms. It uses a protocol suite called "JOSE" (Javascript Object signing and Encryption) to implement smart contracts on the devices [Fil16b]. In order to enable micro transactions on these embedded devices authors of the Filament project propose a solution called Penny Bank. It allows the devices to exchange small amounts of value with each other

without involving the blockchain for every single transaction thereby avoiding heavy transaction fees [Fil16a].

3.3 Supply Chain Management

Blockchains allow secure and permanent documentation of transactions in a decentralized ledger. They can be transparently monitored by all parties. This can improve efficiency and reduce human mistakes and delays. It can also enable users to verify authenticity of products by tracking them from their origin. An example of this is securing supply chains of diamonds from mine to consumers. IBM's Hyperledger Fabric is one of the proposals working in this field. It is permissioned blockchain infrastructure designed specifically for handling supply chain management tasks. Using blockchain technologies customers can verify when and where a diamond was mined, all the places it passed through during its journey to the retailer, and whether or not during any step of the supply chain it crossed any moral or legal grey areas i.e. if it's a blood diamond [IBM17].

3.3.1 Skuchain

Sku chain is a platform that uses blockchain to provide security, efficiency and transparency to supply chains. Today's supply chain management tools such as ERP systems, Inventory Management systems, Letters of credits, Purchase and Invoicing tools have significant friction and problems interfacing with each other. This results in increased costs and delays in every process of the supply chain. Skuchain proposes tools in order to resolve some of these issues [Sku17].

IMT: *"IMT provides inventory financing that de-risks transactions and unlocks capital opportunities for the entire supply chain. The original contract between the buyer and seller is assigned to IMT in the blockchain. This acts as a Blockchain Based Security Interest that provides the collateral to an investor in the IMT fund. IMT uses its funds to purchase goods from the seller and stores them at a VMI warehouse. Finished goods are shipped pursuant to a purchase order from the buyer, a process covered by insurance. The buyer then pays IMT for the goods"* [Sku17].

Brackets: Smart contracts on the skuchain are called brackets. They are cryptographically secured. They provide some key advantages.

- They are designed to release collateral as a result of being triggered automatically by real world events [Sku17].
- They improve transparency for all participants by providing real time view of transaction state [Sku17].

- *"It enhances liquidity of collateralized assets in a supply chain by improving upon current trade finance instruments such as Factoring, PO Financing and Vendor Managed Inventory Financing. It also creates the opportunity for Deep Tier Financing"* [Sku17].

PopCodes: *"Popcodes are Proof of Provenance codes, a crypto-serialization solution to track flow of goods on SKU level. They provide bank-grade traceability to track physical value in the supply chain. Popcodes are sophisticated in their ability to track sub-assemblies, parts and raw materials used to make a finished product. Using Popcodes, an enterprise can gain JIT visibility across the entire supply chain ecosystem, enabling optimal agility and planning. It also provides end-consumer visibility into the entire history of the product"* [Sku17].

3.3.2 Provenance

This project is working on using blockchain technology to enable secure traceability of certifications and other salient information in the supply chain. It aims to become a platform for verifying authenticity of goods. *"Provenance enables every physical product to come with a digital 'passport' that proves authenticity (Is this product what it claims to be?) and origin (Where does this product come from?), creating an auditable record of the journey behind all physical products"* [Ben15]. They are creating a decentralized app for solving certification and chain of custody challenge in sustainable supply chains. It proposes a system to assign and verify certain properties of physical products using the blockchain. There are six different actors involved in the proposed scheme namely: Producers, Manufacturers, Registrars (they provide unique identity to other actors in the scheme), Standards organizations (define the rules of a certain scheme (e.g., Fairtrade)), Certifiers and auditors, and Customers.

The architecture presented in the white paper [Ben15] consists of a number of modular programs. Namely Registration program, Standards programs, Production programs, and Manufacturing programs. Each contract is deployed on the blockchain independently but since all of them work within the same blockchain system they can interact without friction. Technologies such as NFC, RFID, barcodes, and digital tags link physical products to their digital representation on the blockchain. Furthermore, user facing application in the form of smart phone applications will facilitate access to the blockchain. They will aggregate and display information to customers in real time, detailing every step of the supply chain [Ben15].

3.4 Filesharing

Filecoin, SiaCoin and Storj are some of the proposals for creating a decentralized platform for filesharing, storage and cloud computing using the blockchain. The idea is simple users instead of uploading files to a central cloud server hosted at google, Microsoft or Dropbox files are shredded, encrypted and spread across the distributed file storage network based on the blockchain. Only the uploader holds the keys to

call smart contracts to decrypt and reassemble the files. People participating as hosts in the network rent out their storage spaces and get paid in return for the services they provide [mis17].

3.4.1 FileCoin

Filecoin *"Filecoin is a decentralized storage network that is auditable and publically verifiable. Clients pay miners for data storage and retrieval. Clients offer data storage and disk space in exchange for payments. The network achieves robustness by replicating and dispersing content while automatically detecting and repairing replica failures"* [Pro17].

Proof of Storage Proof of storage is a class of decentralized challenge response protocols. They allow a storage provider or host to efficiently verify the integrity of the data stored on their device to their users or clients. The client sends encrypted version of its data to the hosting node for storage, while keeping a small portion of that data himself so he can cryptographically verify the integrity of data stored on the hosting node at any time [Pro17].

Proof of Replication Proof of Replication Filecoin introduced a special form of proof of storage protocols called Proof of Replication. It is an extension of Proof of Storage protocol. It enables a miner to convince a user that some data D has been successfully replicated to its own unique physical storage S. It uses challenge/ response protocol to achieve this [Pro17]. Traditional PoS protocols are limited in that they only prove that a miner or host was in possession of data at the time of challenge/response. Filecoin developed PoR protocol in order to provide stronger guarantees against Sybil attacks, Outsourcing attacks and Generation attacks [Pro17]. These attacks are exploited by malicious nodes to gain reward for storage that they are not actually providing. Such greedy miners reduce the overall capacity and performance of the network. These attacks are discussed below.

Sybil Attacks A malicious attacker may want to claim the reward for storing multiple copies of some data D. They can cheat the system in traditional PoS protocols by claiming to store multiple copies using Sybil identities, while in reality only storing one physical copy of the D [Pro17].

Outsourcing Attacks A malicious miner could exploit the system by quickly fetching the data D he is claiming to store from other nodes at the time of challenge/response [Pro17].

Generation Attacks A malicious miner could rely on small but efficient program to quickly generate the data D when it is requested. This could allow such an attacker to gain reward for storing large amounts of data even when he physically does not have the capacity to do so [Pro17].

Proof of Space Time Proof of Space Time is a novel implementation of PoS. It allows a user to verify that the data was being stored by the miners throughout a period of time. Proof of space time requires a storage provider to produce a sequential proof of storage for a period of time, it then recursively composes them together to generate a complete proof [Pro17].

4 Design Decisions and Technology Stacks

This chapter deals with fundamental concepts and technology stacks used for developing our decentralized supply chain management system. This system is described in chapter 5. The backbone of our system is a smart contract running on Ethereum. In addition to tracking goods, our system is also used for managing costs and paying suppliers for their services. This system employs Raiden Network 4.4 based payment channels to securely handle financial transactions. InterPlanetary File System is used for storing large documents and complete logs related to individual supply chain processes.

4.1 Ethereum

Ethereum is a decentralized blockchain based computing platform that provides smart contract functionality. It aims to be a platform for building decentralized applications (Dapps). Ethereum has a decentralized Turing Complete virtual machine which serves as an abstract foundation layer for building Dapps and running smart contracts. The Ethereum Virtual Machine (EVM) is programmed using a java script like programming language called Solidity.

Motivation

Ethereum was selected as it is the largest and most secure platform for smart contracts and decentralized applications available as of the writing of this thesis. Its open source nature and other design philosophies like Simplicity, Universality, Modularity, Agility and Non-Censorship were also important considerations in favor of its selection [Vit15a]. Ethereum is special because each block in its blockchain represents a state of its virtual machine [Vit15a]. The design philosophy of Ethereum follows a set of principles outlined below:

Simplicity: The Ethereum Protocol is designed to be as simple as possible to fully realize the unprecedented democratization potential of blockchains [Vit15a].

Universality: Ethereum provides a Turning Complete scripting language. Programmers can use this language to develop any smart contract, which can be mathematically defined. Ethereum does not have "features", instead its design philosophy follows the principal: If it can be correctly defined mathematically, then it can be built on Ethereum [Vit15a].

Modularity: The Ethereum protocol is designed to be as modular and separable as possible. This insures that small changes to the protocol do not affect applications that are already running on Ethereum. That is why Ethereum mining algorithm and consensus mechanisms are designed as separate modules [Vit15a]. This is one of the reasons that Ethereum is perfectly poised to adequately meet the scaling challenges facing public blockchains. Designers are already working on multiple scaling solutions including Plasma, Sharding, and Raiden.

Agility: The Ethereum protocol is designed to be modifiable in order to meet the security and scalability challenges of the future. Though designers and developers are extremely judicious about making changes to high-level constructs. However, if during testing and development process developers discover that changes to protocol architecture or EVM may lead to substantial improvements, they are quick to exploit them [Vit15a].

Non-Censorship: The protocol is designed to be application agnostic. It does not restrict specific categories of usage. All regulatory mechanisms are designed to prevent harm to the network as opposed to restrict any undesirable application [Vit15a].

Architecture Elements of Ethereum

4.1.1 Ethereum virtual machine

The Ethereum Virtual Machine presents an abstract architecture to run smart contracts on different mining hardware setups all over the world. EVM serves as the run time environment for smart contracts on Ethereum. It prevents programs from accessing each other's state and ensures that communication can be established without any interference. It is completely isolated from the rest of the main network and as such can be used as a perfect testing environment [JP 15].

4.1.2 Merkle Trees in Ethereum

Merkle Trees are an important part of any blockchains as described in 2.5. The purpose of Merkle Trees is to efficiently verify the data in a block without creating giant block headers, which would have adverse effects on blockchain scalability and decentralization. In Bitcoin each block header contains one tree structure to represent transactions [Vit15b]. Ethereum, on the other hand uses Merkle trees (see fig 8) to represent three kinds of objects [Vit15b]:

- Transactions
- Receipts
- State

Each tree in Ethereum handles a specific type of queries. A Transaction tree is used to verify if a particular transaction has been included in a block or not. A Receipt tree handles events. It can be used to get all instances of a particular event X emitted by a particular address over a period of time for e.g. last 30 days. A State tree is used to confirm validity and existence of an account and its balance. These trees function in a highly efficient manner, in most cases it is just a matter of fetching the correct Merkle branch

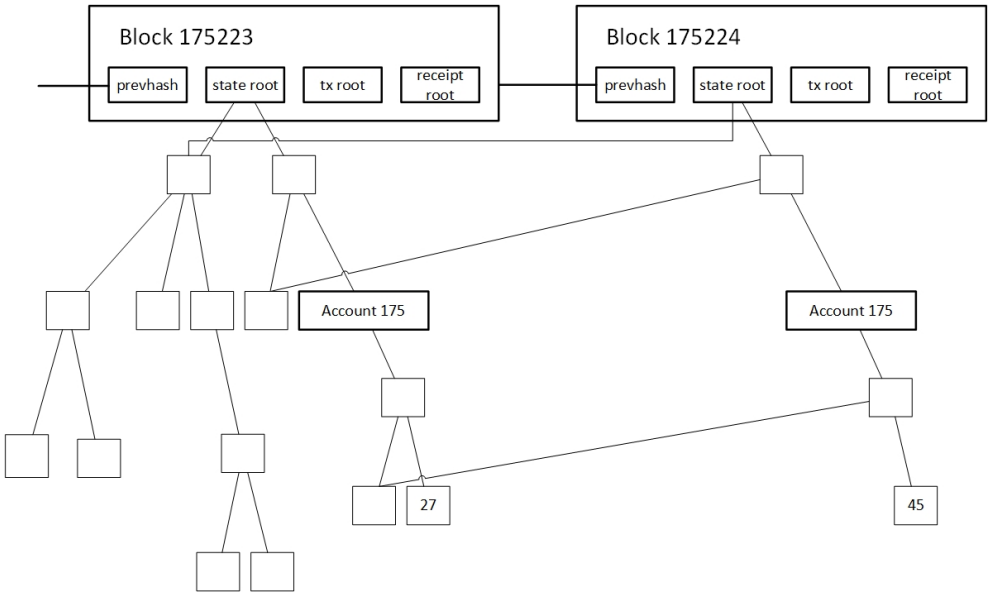

Figure 8: Merkle Trees in Ethereum [Vit15b]

and replying back to the client. Together these three trees allow for the creation of a highly advanced yet light protocol which can be executed by light clients to create a highly decentralized computing network [Vit15b].

Ethereum uses a special form of Merkle tree called Patricia trees [Eth17b]. These are more complex data structures compared to the binary trees used in Bitcoin. Binary Merkle trees are a good data structure for creating transaction trees as the tree is created once and then frozen solid. The state in Ethereum however needs to be frequently updated. This requires a data structure where the root can be quickly calculated after each insert, edit, update or delete operation without reconstructing the entire tree. The state in Ethereum is basically a key – value mapping where the key is the account address and value is a combination of account balance, nonce, code and storage for each address. Storage itself is a tree as well [Vit15b].The Patricia tree is the most suitable data structure for the Ethereum protocol. Detailed specifications for these trees is presented in [Eth17b]. *"The simplest explanation for how it works is that the key under which a value is stored is encoded into the path that you have to take down the tree. Each node has 16 children, so the path is determined by hex encoding: for example, the key dog hex encoded is 6 4 6 15 6 7, so you would start with the root, go down the 6th child, then the fourth, and so forth until you reach the end"* [Vit15b]. Patricia trees have highly desirable secondary properties as well:

- They have bounded depth even in the presence of an attacker, who is deliberately trying to craft transactions to make the tree as deep as possible. This prevents denial of service attacks, which would be possible if each individual update became extremely slow due to large size of the tree [Vit15b].

- The root of the tree only depends upon the data in the tree and not the orders in which updates are made [Vit15b].

4.1.3 Ethereum State Transition Function

Technically any blockchain based ledger can be thought of as state transition function. Ethereum State Transition Function APPLY(S,TX) → S' represented by the figure 9 works as follows [Vit15a]

1. Check if the signatures are valid and the nonce matches with senders nonce [Vit15a].

2. Calculate Gas to pay for the transaction, subtract the gas price from senders account and increment senders nonce i.e. transaction count for senders account [Vit15a].

3. Initialize gas and pay for transaction by taking off certain quantity of gas per byte for each byte in the transaction [Vit15a].

4. Execute the transaction, if transaction is transfer function then move coins from senders account to receiver account. If the transaction is to a contract address, execute the contract code to completion or until execution runs out of gas [Vit15a].

5. If the transaction failed due to not having enough coins or code execution running out of gas revert all state changes [Vit15a].

6. Revert any remaining gas to the senders account [Vit15a].

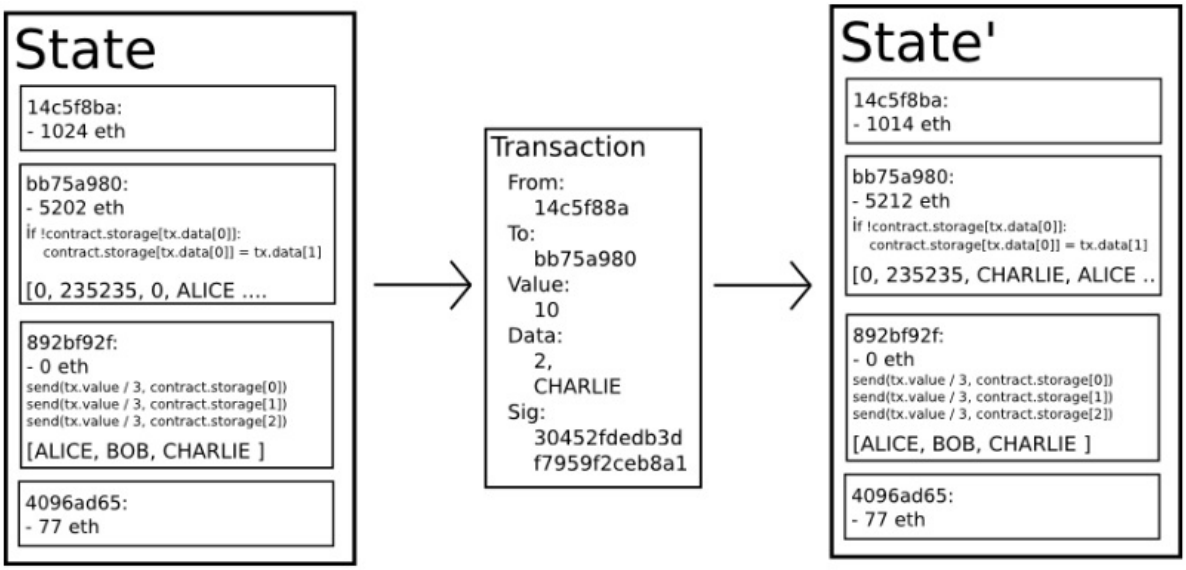

Figure 9: Ethereum State Transition Function [Vit15a]

4.1.4 Block limits and Gas

Block Limits and Gas are regulatory mechanisms to prevent network abuse. These measures are designed to restrict and discourage attack vectors, without censoring nodes and users.

Gas Transactions on Ethereum network cost Gas. Users pay for each transaction in Eth, which is the native currency of the Ethereum protocol. Each operation on Ethereum costs different amounts of gas to execute. The gas price is directly related to the computational complexity of the operation to be performed. Every transaction includes two parameters 'Gas Limit' and 'Gas Price'. The gas limit is the maximum gas a user is willing to pay for executing a transaction, and gas price is the price of basic unit of gas called 'wei' in Eth. The total cost of a transaction in Eth is given by the following equation: [Vit15a]

$$TotalTransactionCost = GasUsed * GasPrice$$

Block Limits The block limit determines how many transactions or operations can fit into any one block. It puts an upper limit on the maximum amount of gas per block. If the block limit is 100 gwei and average transaction costs 20 gwei than a total 5 transactions can fit into a single block. Block limits are created to mitigate infinite loop attacks and malicious denial of service (DoS) attacks. A malicious DoS happens when an attacker creates multiple transactions that are cheap to add to the network but have operations that are computationally difficult to execute for clients. This results in network slowing down due to many pending transactions and consistently full blocks [Hud17]. Block limits are configurable, Ethereum protocol provides a built in mechanism for changing block limits where by miners can vote on the new block limits. This allows to increase the capacity without requiring a hard fork [Vit15a].

4.2 Smart Contracts

Smart Contracts are computer programs that autonomously executes a set of functions or events based on predefined conditions. They allow automatic exchange of goods and services be it money, property, shares or anything of value in a conflict-free way avoiding middlemen. Smart Contracts can be used in all sorts of scenarios like financial services, crowd funding (ICOs), credit enforcements etc [Pet15]. A generic Smart Contract can be represented by a transition diagram illustrated in figure 10. This figure can be explained with the help of the example given below.

Example: This example details a blockchain based property rental service. The user rents an apartment through a service which uses Ethereum blockchain to facilitate payments and key transfers. Properties are listed on a Smart Contract with rental conditions like duration of stay and price per night already stipulated in the terms of the contract (see fig 10). The tenant can agree to the terms by paying for the duration of his stay in cryptocurrencies. The payment will trigger associated events in the smart contract (see fig 10). Funds are blocked by the contract until the tenant receives the key. The key can

be transferred digitally or physically. If the key doesn't arrive, the contract releases the funds back to the tenant. If the key arrives, the contract transfers the funds to the owner of the property. The contract is automatically executed and cannot be interfered by either party. The contractual terms are implicitly agreed to by both parties and are witnessed by hundreds of people / miners [Blo16b].

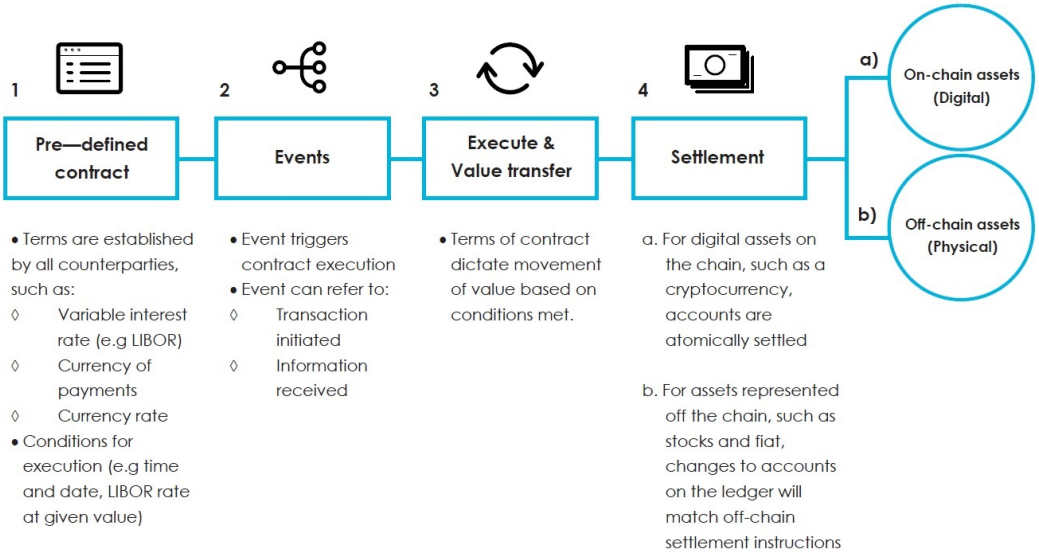

Figure 10: Smart Contract [Pet15]

4.2.1 Interacting with Ethereum Smart Contracts

Smart Contracts on Ethereum are written in Solidity which is a statically typed, contract oriented high-level programming language, developed to target the Ethereum Virtual Machine. Decentralized applications are composed of multiple modules, including modules for interacting with Smart Contracts and other traditional programming functions like GUIs etc. In order to facilitate Dapp development, Ethereum developers created an Application Binary Interface (ABI) to serve as a bridge between Smart Contracts and the rest of the application code. This ABI exposes Smart Contract functions to other application modules, that are developed for traditional environments using common programming languages like JavaScript (Web3), Python (Pyethereum.py), and Go. To facilitate decentralized application development Ethereum provides a number of helper libraries including Web3 for JavaScript and Pyethereum for Python. These helper libraries use the Ethereum ABI to interact with Smart Contracts running on the Ethereum Virtual Machine or EVM. ABI is generated by compiling Solidity code.

Raiden network aims to solve the scaling problem described in 2.7 for the Ethereum blockchain. Raiden was chosen as it was the most stable off chain payment solution available at the time of writing. It uses Payment channels and payment routing to perform ERC20 token transfers in an off chain manner. Payment Routing is a technique to transfer token between two participants who do not have direct connection or channel with each other. This technique advocates using a network of interconnected channels and multi hop transfers to find a path between the participants. An abstract representation of a bidirectional Raiden payment channel is shown in figure 11.

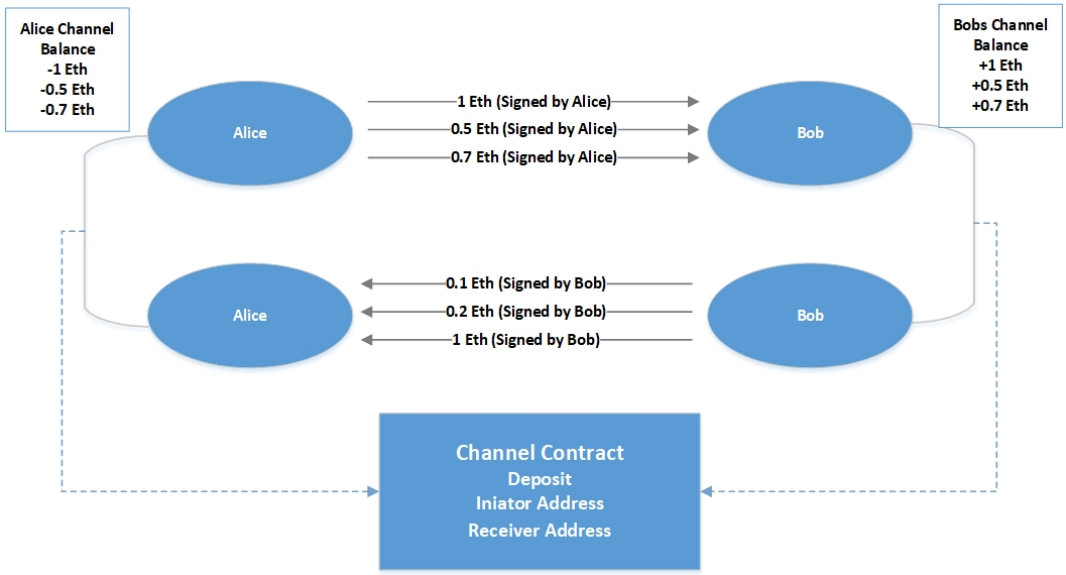

Figure 11: Bidirectional Payment Channel

This figure shows a direct channel between Alice and Bob. The channel can be opened by any participant by deploying the channel contract (see 4.3.1) on the Ethereum blockchain. The channel contract contains a deposit, and two Ethereum addresses i.e. Receiver address and Iniator address. Once the channel is established participants can make multiple transfers to each other by signing transactions with in the channel. Each signed transaction updates the balance of the channel participants as shown in figure 11. The channel can be closed at any point by either participant by putting the last signed transaction of the counter party on the blockchain. The channel contract then executes on the blockchain, it closes the channel and updates the final balance of the participants [Rai17]. This was an example of direct transfer between two participants, however in order for the network to scale it needs to provide a method for transferring coins between participants who do not have a direct channel with each other. Raiden accomplishes this by using so called Mediated transfers shown in figure 12. Mediated transfers can be used in our system to pay shippers by initiating a mediated transfer using suppliers. Mediated transfers are completed using Payement Routing (see 4.3.4). Payment Routing relies on Hash locked transfers describe in 2.7.2 to transfer coins between participants who do not have a direction connection with each other in a trustless manner. In the example shown in figure 12 Alice can transfer coins to Trudy even though she does not have a direct channel with her. This is accomplished by leveraging a

path through the Raiden Network relying on intermediate connections that facilitate multi hop transfers [Rai17]. Sections 4.3.1, 4.3.2, 4.3.3, 4.3.4, and 4.3.5 briefly explain important modules and building blocks of the Raiden Network.

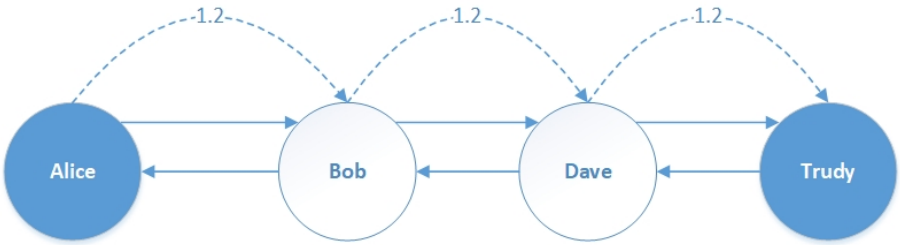

Figure 12: Payment Routing

4.3.1 Netting Channel Smart contract

Netting Channel smart contract is Raiden's implementation of a Payment Channel contract. This contract gets deployed on the Ethereum Block chain when a channel is created. It carries shared rules implicitly agreed upon by the participants of an off-chain channel. It allows for the following: [Rai17]

- Arbitrary number of token/asset transfers between channel participants [Rai17].
- Conditional value transfers that have an expiration and predefined rule to withdraw [Rai17].
- Rules to determine order of transfers [Rai17].

Each netting channel is associated with one bi-directional payment channel. Each channel deals with an ERC20 token and has its own settlement period measured in terms of blocks. ERC20 is a token standard for defining custom tokens or coins on top of Ethereum blockchain. Participants can deposit the associated type of ERC20 token any number of times at any time after the channel is opened. The token transfers are represented by lock structures that contain token amount, expiration and hash lock. The set of pending transfers are encoding into a Markel tree and represented in each transfer by its root. The total channel capacity is equal to total amount of tokens deposited by both participants. Any participant can increase the capacity at any time by depositing more tokens into the channel. The capacity is divided into available and locked balance to each participant/direction [Rai17].

4.3.2 Channel Life Cycle

A Payment channel in Raiden can be in one of the following stages of its lifecycle.

- Deployement
- Funding / Usage
- Close
- Settle

Deployement: A channel is deployed by calling the textit/channel endpoint of Raiden Rest API with the correct ERC20 token address and the partner address with whom the channel needs to be created [Rai17].

Funding: A channel can be funded by either or both parties by transferring and locking tokens in to the channel, once tokens have been transferred each participant can perform an arbitrary number of transfers back and forth provided they have the necessary funds to perform the transfer. A channel user can transfer token to another user by calling the textit/transfer end point as described in 4.3.5 [Rai17].

Close: Once either party wants to withdraw tokens or a dispute arises the channel must be closed. This is done by calling the close function. After the close function is called the settlement window opens. Settlement window is the number of blocks participants have to wait before outstanding balance is settled on the blockchain. Within the settlement window both parties must update the counterparty state and withdraw the unlocked locks [Rai17].

Settle: Any participant can close the channel at any point and from that point no more transfers can be made. Once the channel enters the settlement window the partner state can be updated using the textitupdateTransfer() function. After partner state is updated by participant's locks may be withdrawn. A withdraw checks the locks and updates the participants current transferred amount. Once withdraw is called or settlement timeout expires the final tokens locked in the channel are distributed to the partners according to final balance of each party in the channel[Rai17].

4.3.3 Token Transfer Types

Raiden Network supports three types of token transfer within its payment channels. These are Directed Transfer, Mediated Transfer, and Refund Transfer.

Direct Transfer: A direct transfer does not reply on locks and is automatically completed as soon as the sending party sends the packet containing the signed message authorizing transfer of coins. Trust is assumed between payer and payee but since Raiden operates on top of an unreliable and asynchronous network hence the sender is also agreeing to unconditionally transfer the token. Sender assumes that transfer is complete after sending the transfer packet. The counterparty sends acknowledgment on receiving the message but due to the nature of the underlying network infrastructure either the sending message or the acknowledgement can get lost [Rai17].

Mediated Transfer: A Mediated transfer is basically a hash locked transfer. The lock has an amount that needs to be transferred, a hash lock to verify the secret that unlock the amount and a validity period [Rai17]. Mediated transfers have a sender address, a receiver address and a parameter to determine the number of hops between the two. The number of hops can be zero for direct transfers [Rai17]. Mediated transfers can be broken down into a series of steps given below:

- Alice signs the transfer and sends it to Bob [Rai17].
- when Bob receives the transfer he sends a request for the secret to unlock the transfer [Rai17].
- On receiving Bobs request Alice sends the reveal secret message [Rai17].
- Bob sends an acknowledgement to Alice that he has received the funds. This acknowledgment also serves to synchronize the channel state between the two participants [Rai17].
- Finally, Alice send a secret message to Bob informing him that lock will be removed from the merkle tree and the balance values are updated [Rai17].

Refund Transfers: A refund transfer is a special Mediated transfer that is done when one of the nodes along the hops cannot make any forward progress, and routing backtrack must be done citerad:001.

4.3.4 Payment Routing

Raiden employs a graph search strategy for routing transfer packets between interconnected nodes. This is implemented as a A^* search on sorted path and uses capacity as a heuristic [Rai17]. Consider the example shown in the figure 13. In this graph each node represents a Raiden node, each edge a channel, arrows represent the direction of transfers, solid lines represent the current search space, dashed lines

represent future search space, and the red lines represents closed or exhausted channels. Each node along the hop makes routing decision locally based on what it thinks will be the shortest path for the remainder of the path. These nodes act as mediators forwarding transfers until target is reached or transfer expires [Rai17].

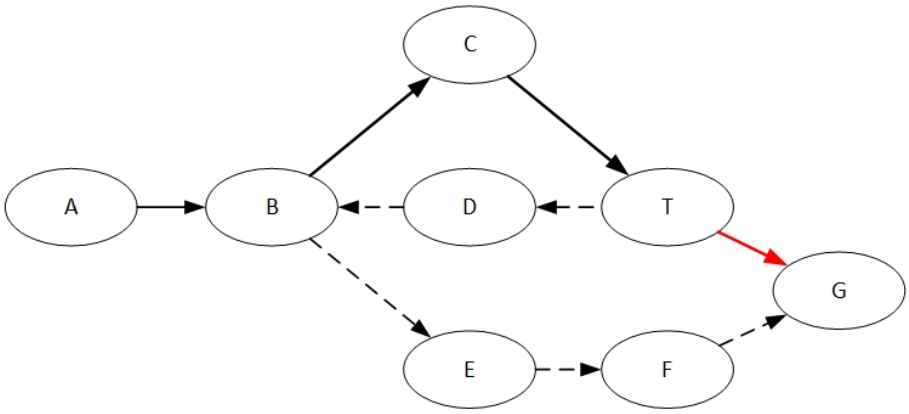

Figure 13: Routing Transfers in Raiden Network [Rai17]

4.3.5 Raiden API

Raiden has a URL encoded RESTFUL API that allows users to interact with Raiden nodes. Raiden accepts and returns JSON-encoded objects. The API path has this format textbf/api/<version>/. Each Raiden endpoint can be interacted with using one of the HTTP methods e.g GET, PUT, POST, PATCH etc. Raiden defines three types of objects: a channel object for uniquely identifying a channel on the Raiden network, an event object for uniquely identifying channel events, and error object for returning errors for any unsuccessful API queries [Rai17]. Detailed documentation about API endpoints is given in [Rai17] however Some important API queries and endpoints are briefly described below:

Registering ERC20 Tokens: A token can be registered by calling the textit/tokens end point with the contract address of the token. Registering deploys a token network contract for that token [Rai17]. This API end point has the following format:

$PUT/api/(version)/tokens/(tokenAddress)$ [Rai17]

Channel Deployment: A channel is deployed by calling the */channels* endpoint of Raiden rest API with the correct ERC20 token address along with the partner address with whom the channel will be established [Rai17]. This API end point has the following format:

$PUT/api/(version)/channels$ [Rai17]

Funding: A channel can be funded by either or both parties by transferring and locking tokens in the channel. Channel is opened by calling the */channels* with channel object as payload [Rai17]. An example of opening a payment channel is given below:

Listing 1: Opening a Payement Channel on Raiden [Rai17]

```
PUT /api/1/channels HTTP/1.1
Host: localhost:5001
Content-Type: application/json

{
    "partner_address": "0x61C808D82A3Ac53231750daDc13c777b59310bD9",
    "token_address": "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
    "total_deposit": 35000000,
    "settle_timeout": 500
}
```

Payment: Either participant can initiate a payment by calling the */payments* endpoint using the http POST request [Rai17]. The API end point has the following format:

$POST/api/(version)/payments/(tokenAddress)/(targetAddress)$ [Rai17]

4.4 InterPlanetary File System (IPFS)

4.4.1 Motivation - Case for decentralized Repository

Supply Chain Management processes generate large amounts of data in the form of shipping manifests, order histories, custom invoices, and shipment logs etc. These logs are vital for documentation, compliance and analytics purposes. They stream line existing processes and help in planning more efficient solutions for the future. These reasons make it necessary to have a complete and in-depth log of every activity in the supply chain cycle. The initial iteration of our Supply Chain Dapp was using Ethereum blockchain to store everything including data, conditions, and logs etc. However, initial testing and evaluation of individual submodules soon revealed that storing large chunks of data on a public blockchain such as Ethereum becomes prohibitively expensive 7.3.1 and hence unsuitable for most companies.

4.4.2 IPFS

The problems mentioned in section 4.4.1 motivated the search for alternate solutions to serve as a decentralized repository for storing data and logs generated by our Dapp. The ideal solution needs to be decentralized so as not to introduce a single point of failure in an otherwise decentralized environment. It should provide easy read access to all participants while simultaneously preventing unauthorized writes or edits. Interplanetary File system or IPFS [Ben14] is a peer to peer hypermedia protocol that aims to be a decentralized replacement for most common Internet protocols like HTTP and FTP etc. These protocols determine the core architecture of most modern centralized applications on the internet. IPFS

satisfies all the requirements described earlier. It is designed as a content addressable decentralized replacement for the web. Content-addressed distributed network such as IPFS has a lot of benefits: It decouples data from servers and replicates them across the network of nodes, Data can be stored close to the user, it allows users to verify the integrity of data in the presence of untrusted hosts, and it also protects against DDoS attacks [Ben14]. IPFS can be seen as single BitTorrent swarm exchanging objects inside a git repository. It provides a high performance content-addressed block storage model with content-addressed hyperlinks for building file systems, decentralized applications, blockchains, and even permanent web pages [Ben14]. Detailed design and description for internal working of IPFS can be found in [Ben14]. A brief summary of important building blocks of IPFS is given below:

Identities: Nodes are identified by their NodeID, which is a hash of their public key. IPFS uses S/Kademlia's static puzzle to generate pair of public and private keys. The keys are encrypted with a passphrase and stored on the node. Nodes exchange public keys with their peers and verify each other identities by calculating hash of the public key and checking if it matches peers NodeID [Ben14].

Network: This module manages connections to other peers using various underlying protocols. It provides a number of desirable primitives to insure smooth functioning of the IPFS network. It uses SCTP to provide reliability even if the underlying network is unreliable. It provides authenticity and integrity of data by using checksums and HMACs. Finally, connectivity in all network environments is insured by employing ICE NAT traversal techniques [Ben14].

Routing: IPFS uses DSHT based routing system to find peers that can serve particular objects. The DHT is created using S/kademlia and coral [Ben14].

Objects: Objects in IPFS are quickly hashed and stored in a Merkle Dag data structure. This insures that objects that contain the same content are stored only once and that all content and their links are uniquely identifiable through content addressing scheme. IPFS uses directed acyclic graphs to store links between objects as cryptographic hashes [Ben14].

Files: Files are stored in a versioned filesystem created on top of Merkle DAG. IPFS file objects are modeled on Git objects and as such are very close to them in design [Ben14].

Naming: IPFS uses a mutable naming system called IPNS to store and address mutable objects and contents. Without IPNS it would be necessary to communicate new hash address with other nodes every time some content was change in a mutable object. IPFS achieves this by defining a mutable state. Every user is assigned a mutable namespace of the form */ipns/NodeID* [Ben14]. Users can publish an object to this path by signing it with their private key. Other users can retrieve the object and verify its integrity by checking signature of the uploader. Mutable objects cannot be content addressed hence IPFS relies on its Routing system to publish their hashes and counts on state distribution through the network for other nodes to find mutable objects [Ben14].

5 Decentralized Supply Chain Management System

The main contribution of this thesis is a decentralized supply chain management and shipment tracker system. This system is proof of concept implementation to explore applications of blockchain for developing decentralized supply chain management and shipment tracking systems. As discussed in chapter 4 the backbone of our system is an Ethereum based smart contract. This contract allows invaluable and real time insight into the state of goods at every step of the delivery process. The logistical data is gathered using IoT based smart packages and communicated directly to the smart contract. The requirements for our system are derived from the use case presented in section 5.1, so as to define realistic goals which could be realized in the given timeframe.

5.1 Use Case Description

Consider an organization that has several suppliers around the world. It uses these suppliers to deliver different components and packages to its factory floor. These packages must be delivered on time and have to be stored, handled and transported under specific environmental or physical constraints. The Organization and Suppliers set up a business contract detailing delivery conditions. The business contract stipulates guarantees about delivery time and the conditions, according to which the package needs to be handled: for example at no point should the package be exposed to temperatures above a certain threshold. This business use case is modeled in figure 14. The handling constraints are instantiated as requirements in the Smart Contract. Once requirements are set the Smart Contract behaves independently from the perspective of the shippers. This is shown in the figure 14 by modeling the two instantiations as independent contracts i.e. Smart contract A.1 and Smart Contract A.2. We will refer to them as SC-A.1 and SC-A from here on. Smart contracts insure trust less compliance of the agreed upon terms and conditions. Payment channels are established between the organization and its suppliers to insure friction less payments and remunerations. In this case two payment channels exist Payment Channel A between organization and Supplier A and Payment Channel B between organization and supplier B as shown in figure 14. These will be referred to as PC-A and PC-B from here on. The system can further be extended to have payment channels between suppliers and shippers as shown in the use case diagram. These are modeled as Payment Channels C and D. These will be referred to as PC-C and PC-D. The organization is responsible for compensating every actor in the supply chain life cycle. In order to do so it can either use Ethereal or issue its own ERC20 tokens which can be exchanged for Eth. The main advantage of issuing your own ERC20 token is that you can fix the exchange rate so it's always tied to a fixed Fiat value. This will protect against price fluctuations in Cryptocurrency value. All payment channels for a particular supply chain cycle use the same token. The suppliers and shippers get payed in ERC20 tokens.

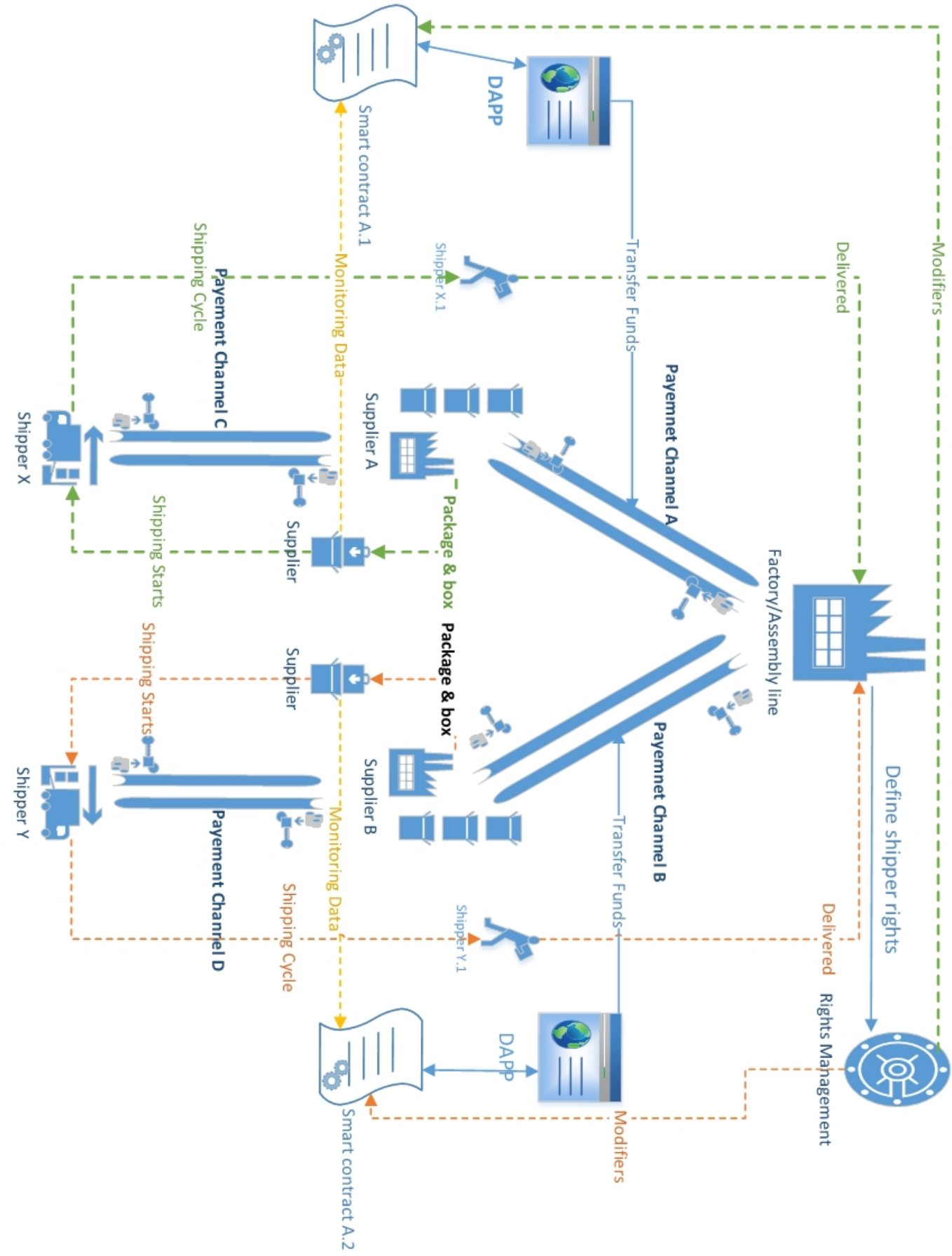

Figure 14: Shipment Tracker - Use case diagram

5.1.1 Supply Chain LifeCycle

In order to better explain the supply chain life cycle of our particular use case we use the simplified version of the system presented in figure 15. The simplified version consists of only one supplier and shipper. The company places a new order for components with its supplier and stipulates shipping and handling constraints in the smart contract.

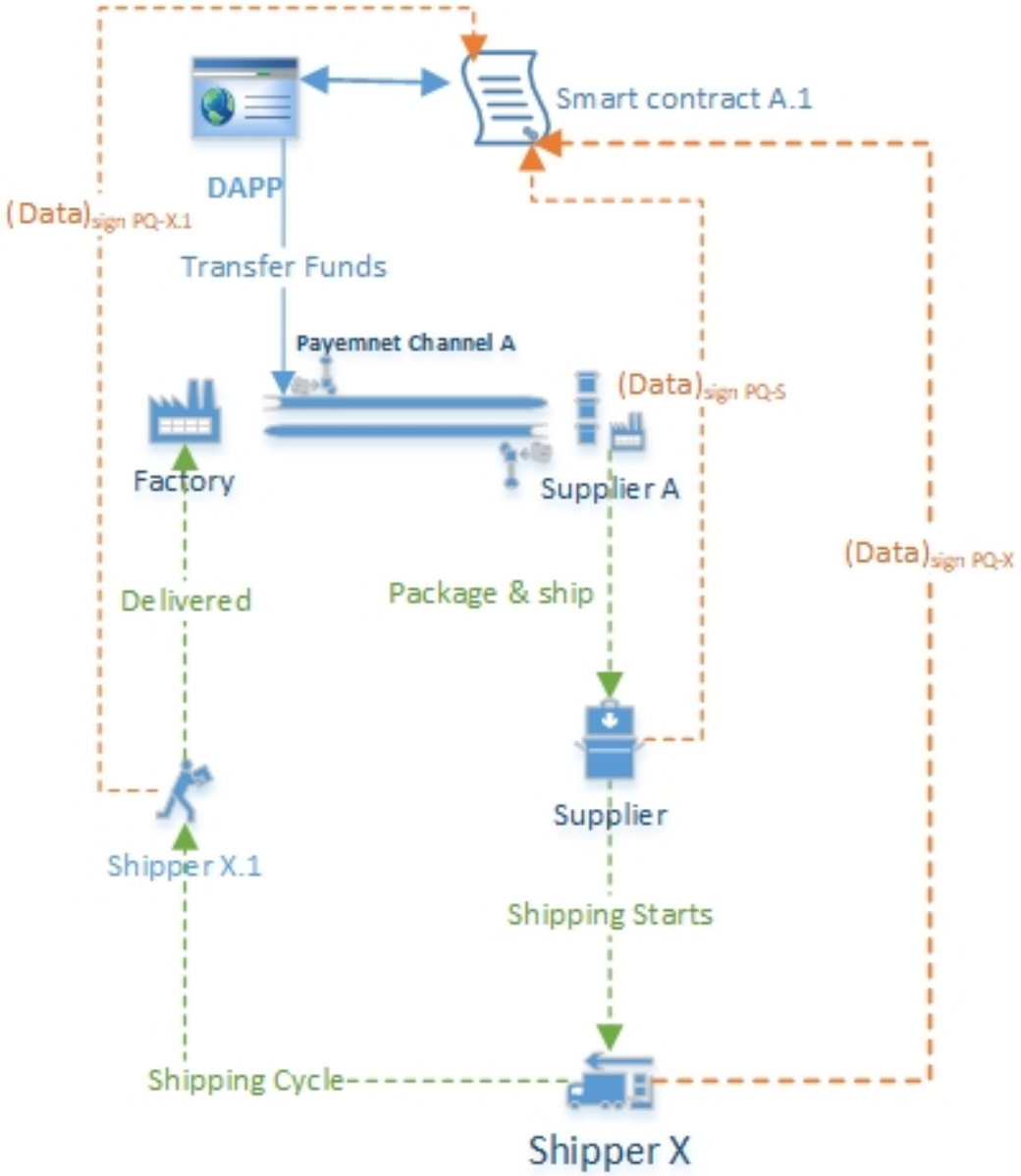

Figure 15: Supply chain Life Cycle

Once the order has been placed it funds the payment channel associated with its supplier. Supplier packages the components along with a tamper proof smart device which will communicate shipping data with the smart contract. The smart device is a Raspberry Pi running a custom program developed as part of this thesis to monitor package conditions and to send signed data to the smart contract.

The data is signed with the help of a lattice based post quantum secure algorithm provided by Dr. Rachid El Bansarkhani. This algorithm protects integrity of data in the presence of post quantum adversaries. The hardware and software that collectively define a smart package will hence forth be referred to as the IoT Node. This node can be configured to send data continuously or when special events are triggered i.e. some shipping violation has occurred. We need internet connectivity throughout the shipping life cycle in order to insure monitoring data is continuously communicated with the Smart Contract and IPFS. The IoT node signs the data with the Post quantum key of the current shipper/ handler. Monitoring starts as soon as the components are packaged in the supplier warehouse, at this time the data sent to the SC-A.1 is signed with key of the supplier. When the shipper X takes possession of the package from the supplier they scan the package. This scanning event represents changing of ownership of responsibility in the supply chain cycle which means from this point on all data sent to the smart contract will be signed with the key of the shipper X. If there is more than one shipper as show in figure 14, Each one scans the package to take over shipping responsibilities. In our system Smart Contracts and Dapps are responsible for catching violations and taking appropriate actions to penalize violators and compensate aggrieved parties. This brings full transparency for all stake holders in the supply chain life cycle and resolved disputes if any in an efficient and trust less manner.

5.2 System Architecture

The proposed system for the use case described in 5.1 has four main components:

1. Ethereum Smart contract
2. Raiden Payment Channels
3. Decentralized Application or Master nodes
4. Smart packages or IoT nodes

The abstract system and processes are modeled in figure 16. Supplier and company agree on shipment conditions in the form of a business contract. These conditions are configured at the start of each shipment cycle. They are instantiated as requirements in a smart contract on the Ethereum blockchain. They cannot be altered after being set in the Smart Contract. The terms of the contract can include any type of handling constraints like exposure to high temperature, pressure, air etc. IoT enabled packages 6.4 monitor the package contents according to the defined constraints and conditions with the help of embedded sensors. They communicate shipping and logistics data with the Smart Contract. The high level functions of IoT Node or Smart Package are modeled in the abstract design diagram shown in figure 16.

The Smart Contract can detect any breaches of the agreed upon terms. If any breaches are detected the system can be used to take appropriate actions to compensate or penalize the parties involved. Payment channels are established between the organization and its suppliers to handle payments efficiently. The payment channels are deployed using Raiden. A company and its suppliers agree upon a set of conditions either implicitly or explicitly at the start of each shipping cycle. The terms of conditions are instantiated in the contract with the help of Contract number or tracking number of the package. The supplier initializes the IoT module inside smart package with the tracking number. Once initialized this module contacts the smart contract to receive set of conditions associated with the tracking number. The IoT device then starts the monitoring procedure as shown in figure 16. In order to minimize transaction costs IoT devices only sends violations to be stored in the blockchain. Full tracking logs containing sensor data like temperature, location, humidity etc are stored using IPFS. The log address or Hash is stored in the blockchain. This enables our Dapp to always have access to full logs. It can query IPFS nodes using the stored hashes to download complete logs related to any shipping cycle. Once the package is delivered the organization can check the Dapp interface to see if any conditions were violated. If no conditions were violated it transfers full amount to the supplier or the shipper. In the event that violations were detected, smart contract communicates them with the monitoring Dapp. If any violations occurred the Dapp can determine the party responsible so that appropriate actions can be taken. These actions could range from giving the supplier or shipper a bad rating to monetary punishments such as withholding payments etc. Penalties are determined based on the severity of the violations.

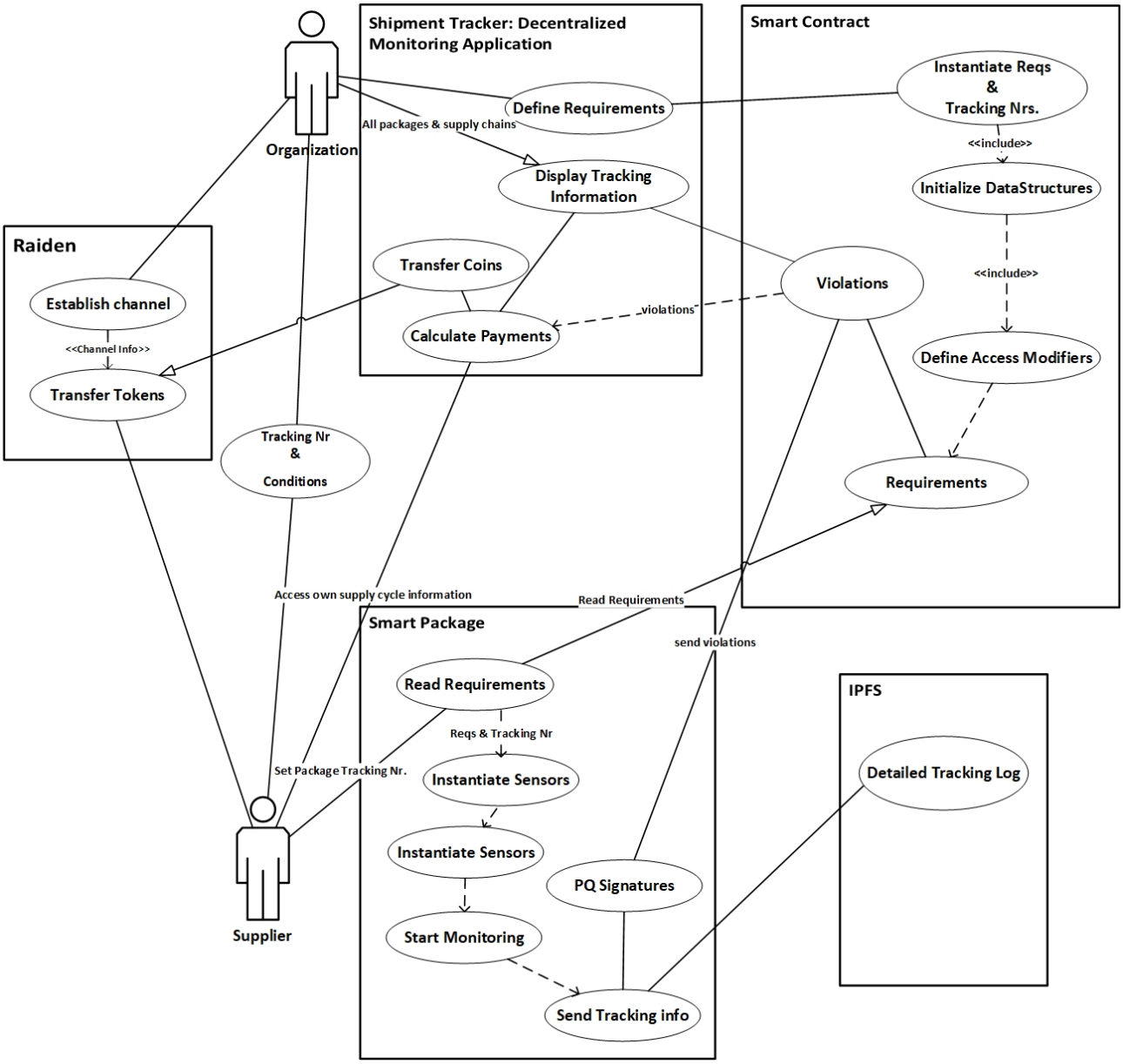

Figure 16: Shipment Tracker – Abstract Design

5.3 Architecture - Block Diagram Overview

Figure 17 shows the architectural overview of our system. This system is composed of three high level components, Master Node (see 5.4.1), IoT Node (see 5.4.2), and Smart Contract (see 5.4.3). The Master Node provides overview of the entire supply chain including activities within individual supply lines. It also handles compensation and payments to suppliers. The IoT Nodes monitor package conditions and communicates logging and tracking data with Smart Contract and IPFS. The Smart Contract provides function and logic to serve as a trustless bridge between Master Node and IoT Node. Master Node queries the smart contract to get IPFS log locations and shipping violations.

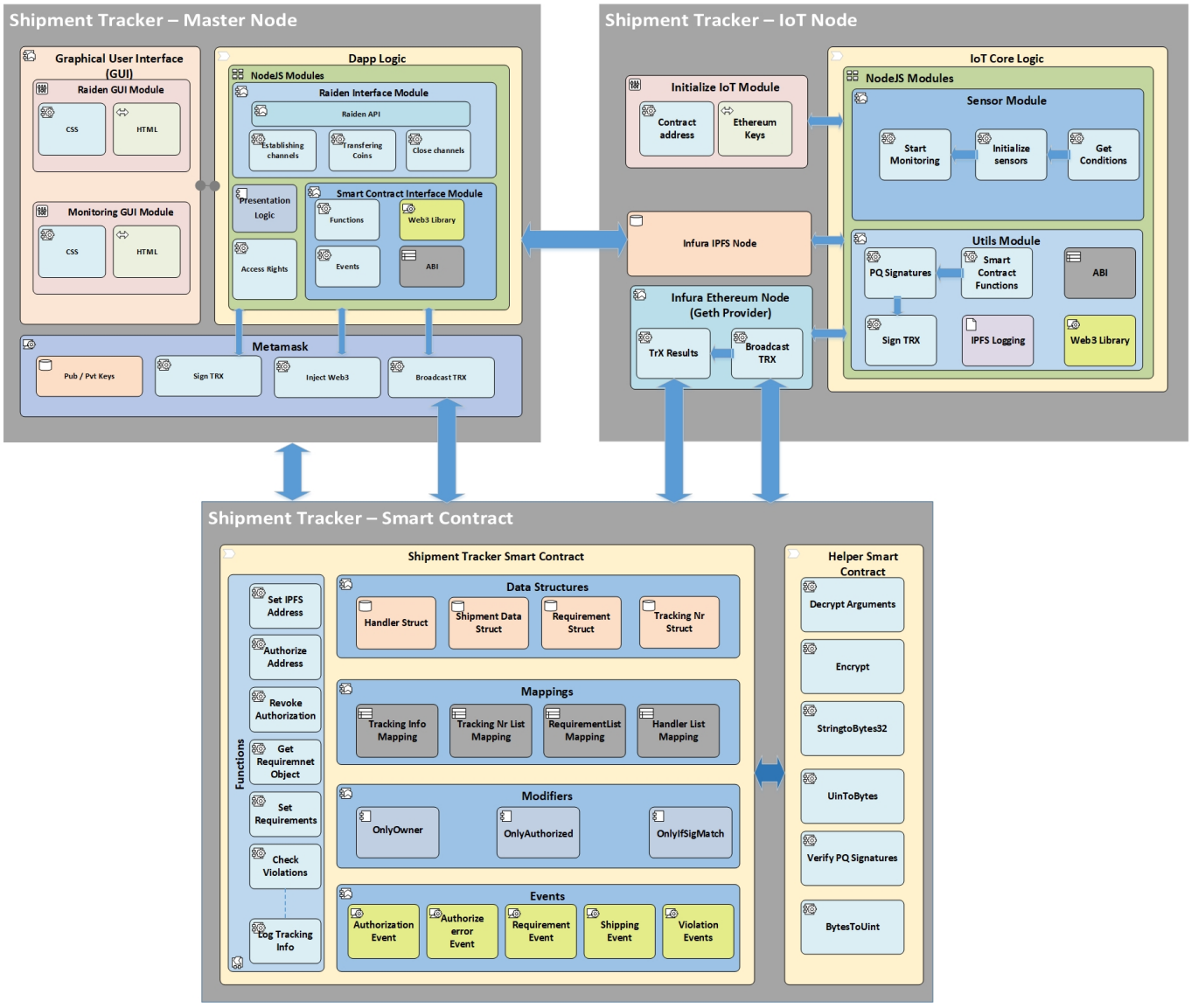

Figure 17: Shipment Tracker – System Block Diagram

5.4 System Components

5.4.1 Decentralized Monitoring Application - Master Node

Figure 18 details the architecture of Master Node. This component contains the main decentralized application for monitoring all shipments across multiple supply lines. Its primary intent is to provide administrators or managers a comprehensive overview of the entire supply chain. Ethereum public key serves as the main access control mechanism in this module. It uses the public key to determine which processes or supply lines a user can view or access. The access rights are defined and stored in the smart contract. Master Node has two main sub modules. One module deals with payments and payments channels. This module relies on Raiden (see 4.3) API to establish payment channels. Raiden client must be running and fully synced with Ethereum blockchain for this module to function. The second module interfaces with the Shipment Tracker smart contract deployed on the Ethereum blockchain. The associated GUI for this module (see fig 18) can be used to define requirements and constraints on shipments and packages that must be followed by suppliers and shippers. This module uses web3 JavaScript library, and MetaMask browser extension to broadcast transactions to the Ethereum Blockchain. MetaMask allows us to run our Dapp Master Node without running a full Ethereum Node. The implementation details and functions of each submodule in Master Node are described in 6.3.

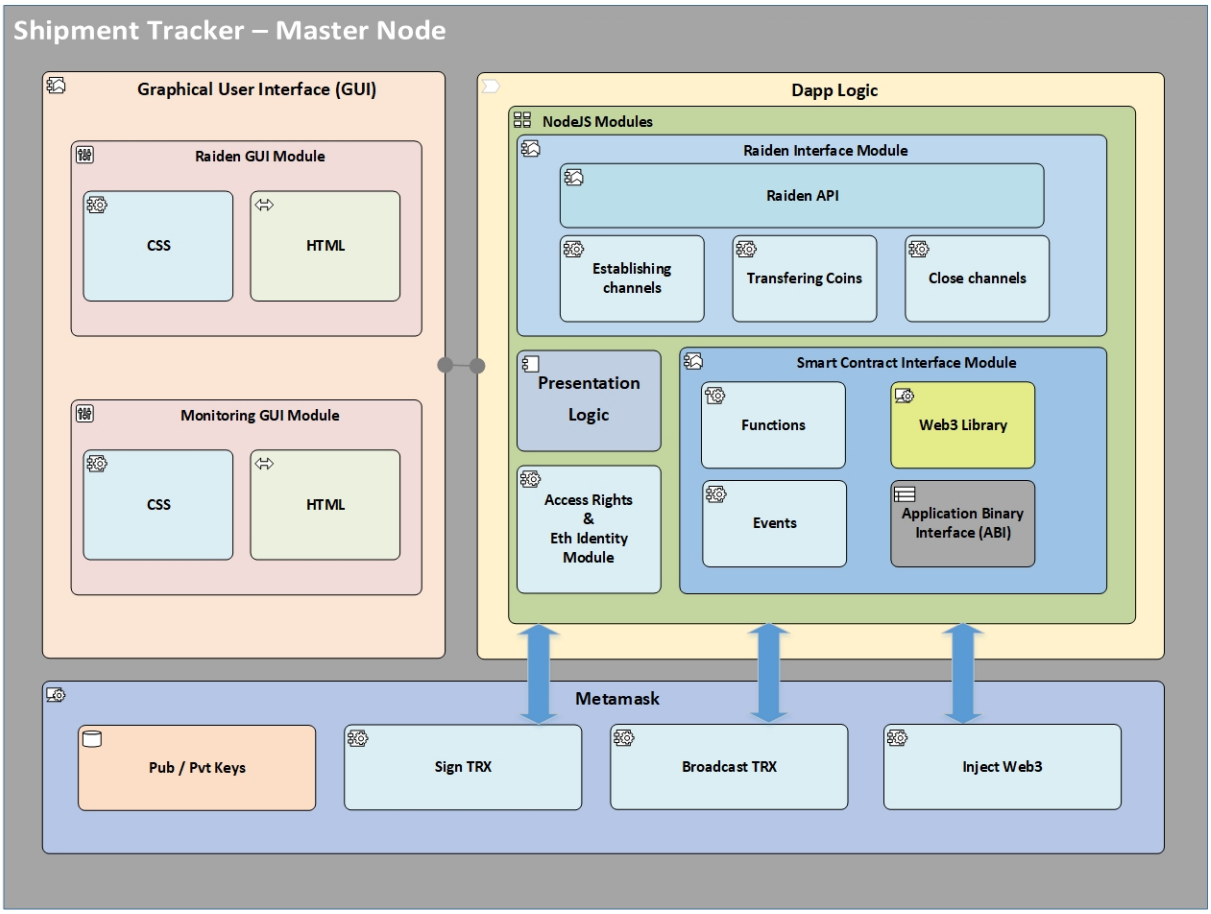

Figure 18: Master Node – Block Diagram

5.4.2 IoT Powered Smart Packages - IoT Nodes

Figure 19 shows the architecture of IoT Nodes. IoT Node is an embedded device with various sensors attached to monitor shipment/package conditions. IoT core logic is sub divided into three main modules Initialization Module, Sensor Module, and Utils Module. Initialization module is responsible for bootstrapping the device at the start of each new shipping cycle. It sets the shipment tracking number, sets the contract address, sets monitoring and logging intervals, and initializes the attached sensors. The device enters monitoring phase after initialization is complete. In this phase the IoT node first contacts the Shipment Tracker Smart Contract to get a list of conditions associated with the current tracking number. It then starts monitoring functions to periodically check that no condition gets violated. If any violations are found it sends them to the Smart Contract to be permanently recorded on the blockchain. In order to communicate with the blockchain we must have access to an Ethereum client. IoT nodes use the remote Ethereum Geth client provided by Infura. The client provided by Infura is just that, unlike MetaMask it does not store keys or handle transaction signing. It expects that any state altering transactions are already signed. IoT Node uses the Utils module to send signed transactions to the Geth client hosted on Infura. The Utils module is also responsible for storing shipping logs on the IPFS Network. The implementation details of the IoT Node are explained in 6.4.

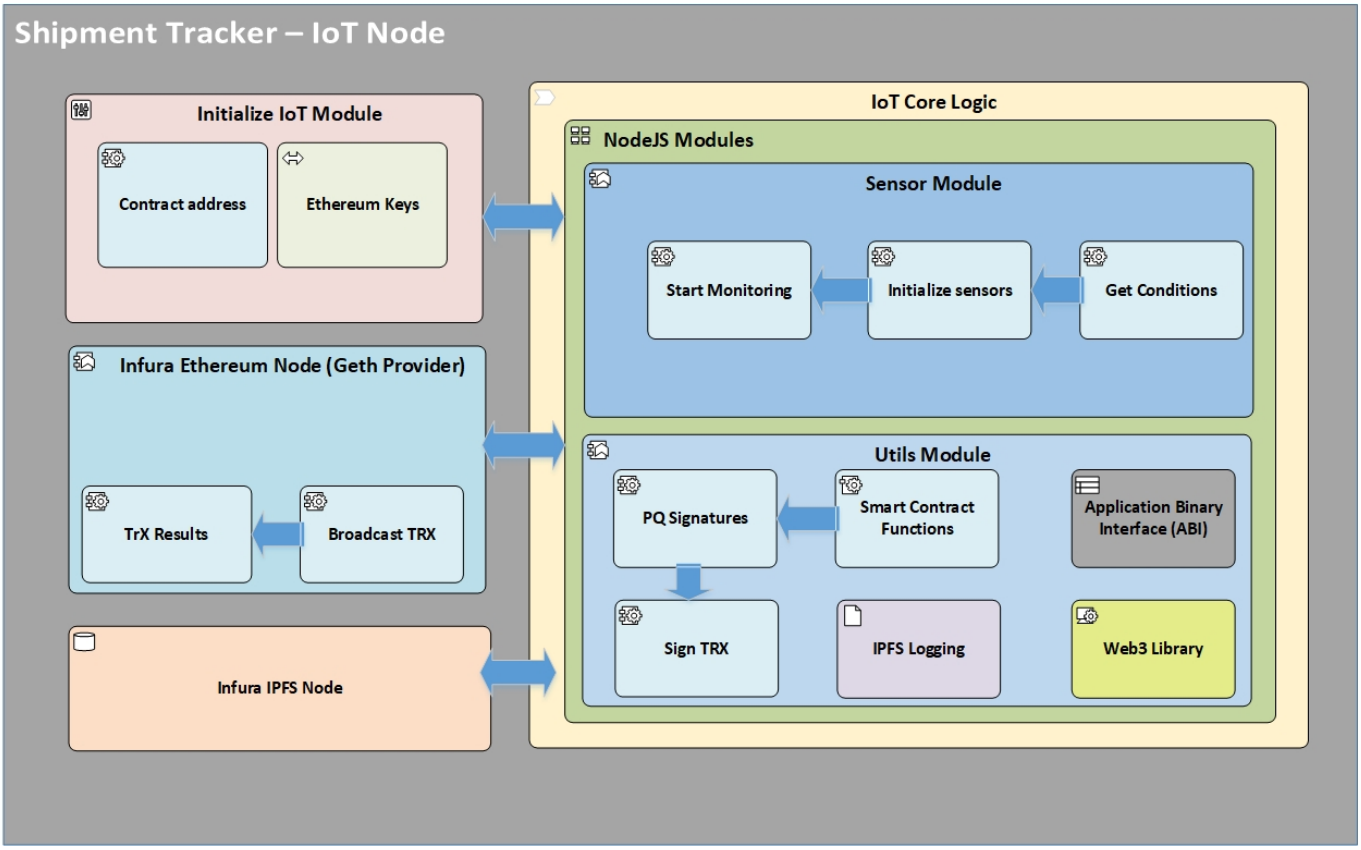

Figure 19: IoT Node – Block Diagram

5.4.3 Shipment Tracker - Smart Contract

The shipment Tracker contract is composed of several components as shown in figure 20. Functional logic of this contract is broken into several public, private and internal functions. Details of some of the important functional component is given in 6.5. The contract defines several data structures to store logical data. Shipment Data and violations sent by the IoT node are stored in ShipmentData (see 6.5) structure, Requirements for individual shipping cycles are held in Requirements (see 6.5) structure, and the list of authorized handlers for a individual shipments are saved in Handler (see 6.5) structure. Modifiers in solidity modify function behavior based on some condition. In our contract modifiers are used for restricting access to certain functions and mappings to only authorized personal or devices. I use the modifier OnlyIfSigMatch to insure that data sent by IoT devices is correctly signed by the Post Quantum key of current shipper. Only verified violations are added to the blockchain. Events are used to inform the Master Node, if any state altering function was executed by the Smart Contract. The final component in figure 20 is the library contract helper.sol. In solidity library contracts are singletons which do not store anything and can be called by any contract to extend their functionality. I developed library contract to handle post quantum signature verification. The main contract calls this library contract to verify PQ signatures. The library driven development helps to make our Smart Contract modular. It also insures that the main contract does not exceed the block limit. The current block limit on Ethereum main net is eighth million gwei.

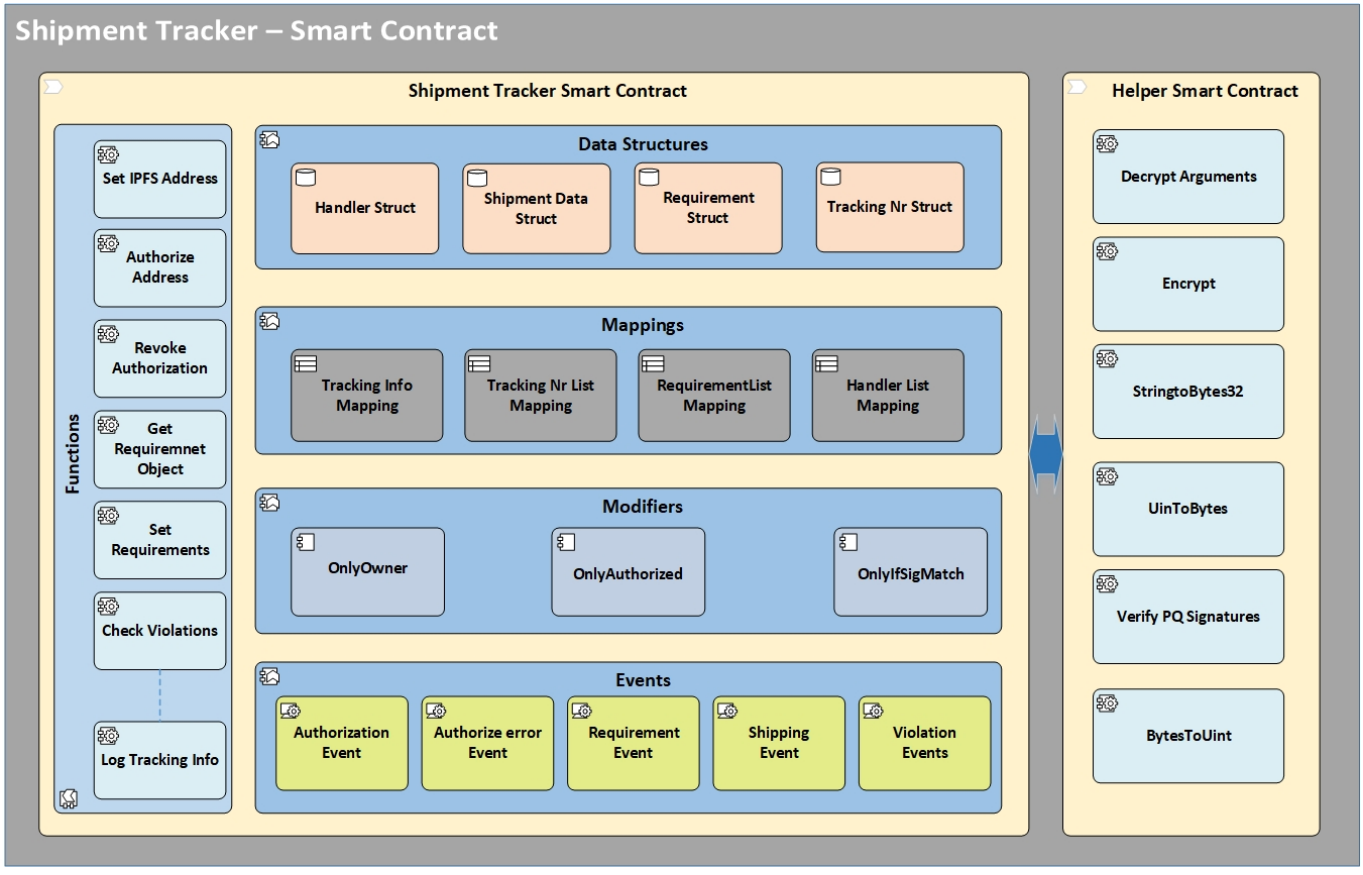

Figure 20: Smart Contract – Block Diagram

6 Implementation Details

6.1 Overview

This chapter describes the technologies and modules, which are created or used during the implementation of the proposed architecture. Our solution is implemented as three distinct high level components. In order to better understand the internal workings of each component, it is helpful to know how these modules interact with each other. These interactions are modeled in the sequence diagram shown in figure 21. The company can use Master Node to track packages in route. Supplier and the company agree on the terms and conditions of shipment in advanced. A representative of the company like an admin then uses the Master Node to set shipping constraints that must be followed by all parties involved in shipping and handling process. These constraints map to a unique tracking number in the smart contract and as such are distinct for each package. The company then funds the payment channel that exists between the supplier and itself. The payment channel should have enough funds to pay the shippers and suppliers upon delivery of the package. Master Node provides functions to issue API calls to Raiden for establishing and managing payment channels. Raiden client must be running and fully synced with Ethereum blockchain for the API calls to be successful. Master Node also provides the functionality to give controlled access to IoT Nodes and shippers so that they can communicate shipping data with the smart contract (see addHandlers(...) in fig 21). This is described in more details in 6.5.2. The supplier is responsible for initializing the IoT device and packaging it with the shipment. IoT node contacts the Smart Contract to receive requirements associated with its tracking number. It then enters monitoring phase. In this phase the IoT Node uses sensors to continuously monitor the package conditions according to the requirements. These conditions may include temperature, humidity, light, and location or some combination of all of them. The IoT Node stops monitoring after the package has been delivered. If no violations happened during shipment the company instructs the Master Node to release funds to its supplier.

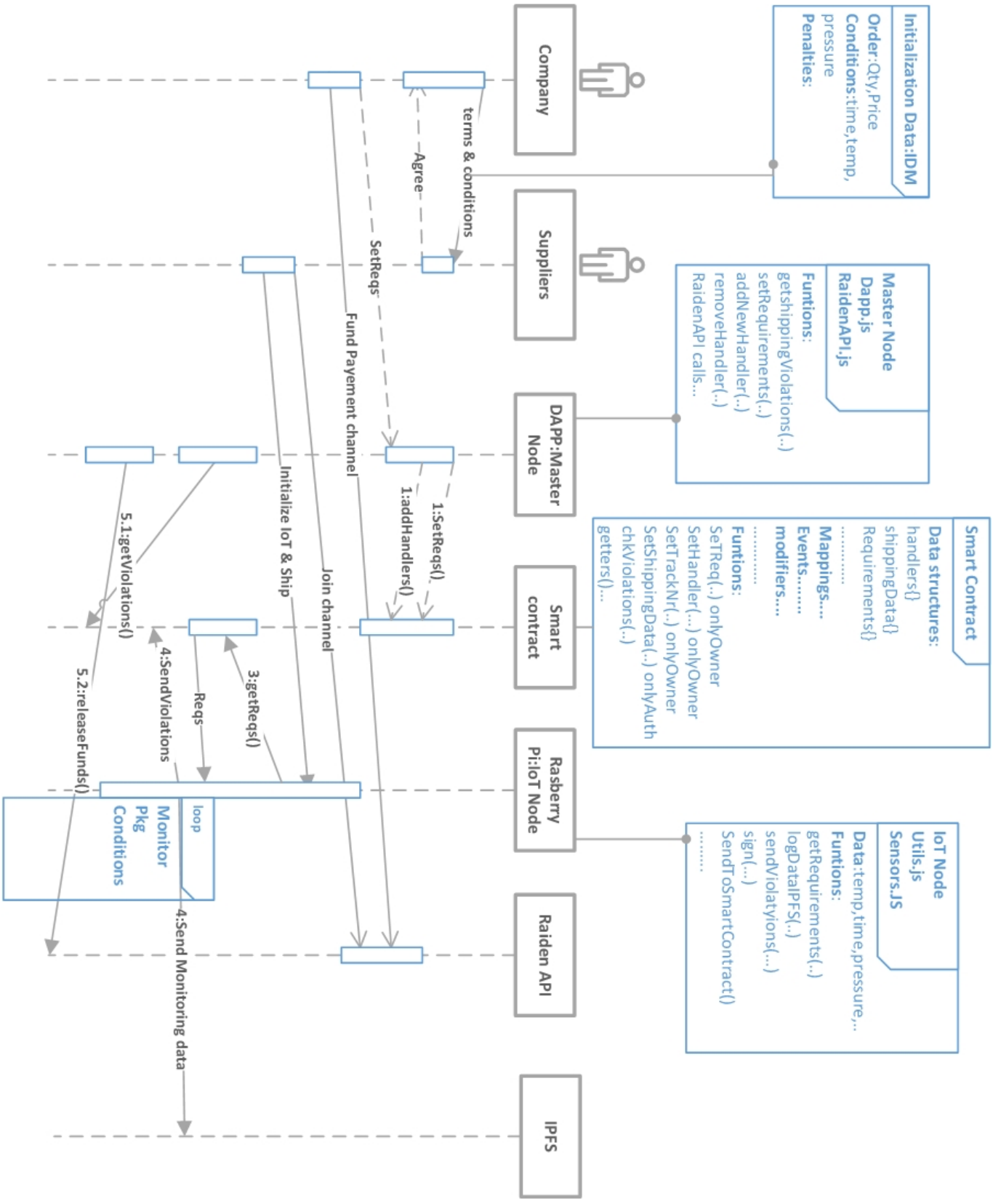

Figure 21: Sequence diagram of the proposed package tracking system

6.2 Hardware Setup

6.2.1 Master Node

Master Node is implemented using NodeJS. All software dependencies are managed by node package manage i.e. NPM. This enables us to run the Master Node on any windows or Linux system capable of running NodeJS. Master Node sub-modules including Raiden and Geth are quite efficient in terms of RAM and CPU requirements. This enables us to run the Master Node on any moderately capable modern machine. As an example Master Node has been tested on a dual core virtual machine with a 4 GB RAM. There are however space constraints (HDD capacity) imposed due to the Raiden Network. Current iteration of Raiden requires that an Ethereum client must be running locally and fully synced with the blockchain. According to block explorer site “etherscan.io”, the current blockchain size of Ethereum main net is 98 GB, and of Ropsten test net is 50GB. Raiden Network team is currently working on an update which will enable users to sync with remote Ethereum nodes such as those provided by Infura. Until this update is ready the Master Node requires at least 60GB of free hard disk space to manage payment channels when deployed on Ropsten test net.

6.2.2 IoT Node

The prototype IoT Node is shown in figure 22. It is implemented using a RaspberryPI 3 model B board, which is running Raspbian Stretch operating system. IoT node runs a custom package monitoring software solution, that is designed using NodeJS. We use GrovePi+ board to interface and connect Temperature, humidity, Light, and GPS sensors with the Raspberry PI.

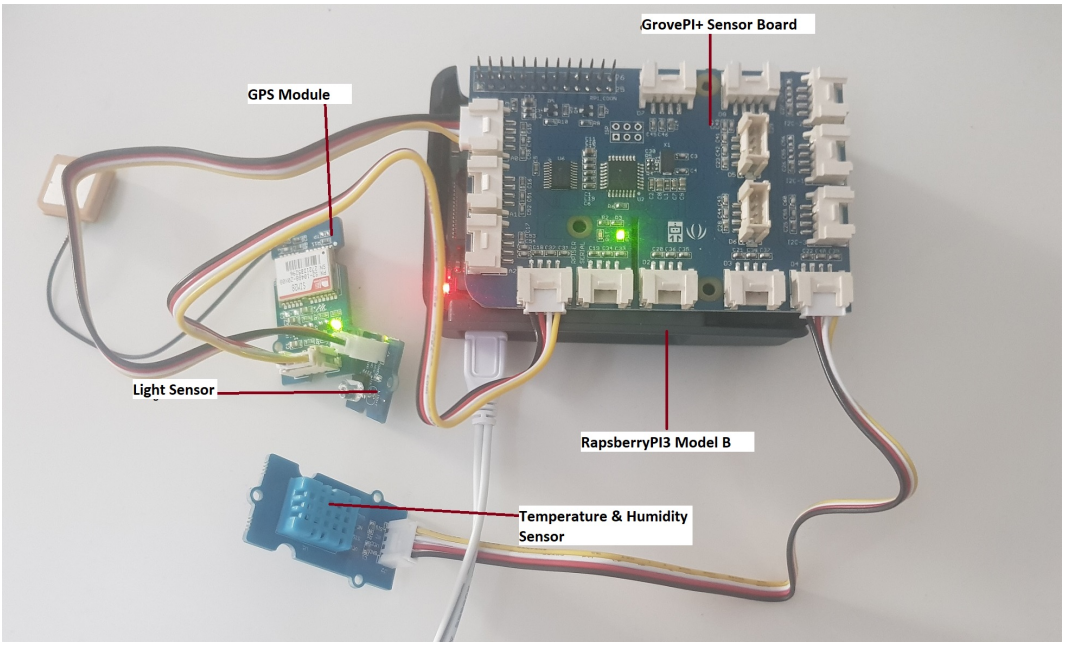

Figure 22: Prototype IoT Node

6.3 Master Node – Implementation

The Mast Node has four sub modules as shown in its work flow diagram detailed in figure 23. These modules are explained below:

login: This is the first page that is presented to the user after starting the Master Node. It asks the user to provide the contract address of the Shipment Tracker contract. It also checks if the provided address is a valid Ethereum address. If the address is valid than Dapp.js module is called.

MetaMask: MetaMask is a browser extension that enables users to communicate with decentralized applications deployed on Ethereum. It eliminates the need for users to run a local Ethereum node. It can be thought of as an Ethereum browser. MetaMask packages an Ethereum wallet, a Web3 injector and an Ethereum client all into one application. It enables our Dapp.js module to send signed transactions to the blockchain. It also manages identity of the user as it securely stores users public and private keys. The Dapp.js module queries MetaMask to get the public key for constant function calls.

Dapp: This module provides the interfacing logic for communicating with the Shipment Tracker contract. Once loaded it first searched for the web3 library which in this case is injected by MetaMask. If Web3 library is not injected, then it looks for web3 and an Ethereum client on the local machine. The user can use the graphical user interface of this module to call functions from our Smart Contract. The contract ABI is packaged and stored with this module. There are two main types of functions in any Smart Contract i.e. constant functions and non-constant functions. The Master Node uses the MetaMask Ethereum client to get results of constant functions. Recall that constant functions do not alter any state in the blockchain. This is why they can be called without incurring any gas or fee. The remote Ethereum node simply executes these functions locally and returns the result to the Dapp module. This is shown in figure 23 with the decision box which checks if the transaction is a call to a constant function. State altering functions must be called through signed transactions. The Dapp module uses Web3 and contract ABI to prepare a transaction payload and then hands it over to the MetaMask. Which then prompts the user to confirm gas payment by signing the transaction. It then proceeds to broadcast the signed transaction to the Ethereum Network. The transaction is then mined and every node on the blockchain executes the Smart Contract function associated with the transaction. **RaidenAPI:** Users can choose to load the RaidenAPI module any time after a successful login. This module presents the user with a unified GUI that shows all open channels. It allows consuming Raiden channels and payments endpoints from the RESTful API. The Raiden client must be running and fully synced for the API calls to be successful. The user is responsible for insuring that the client is running before starting this module.

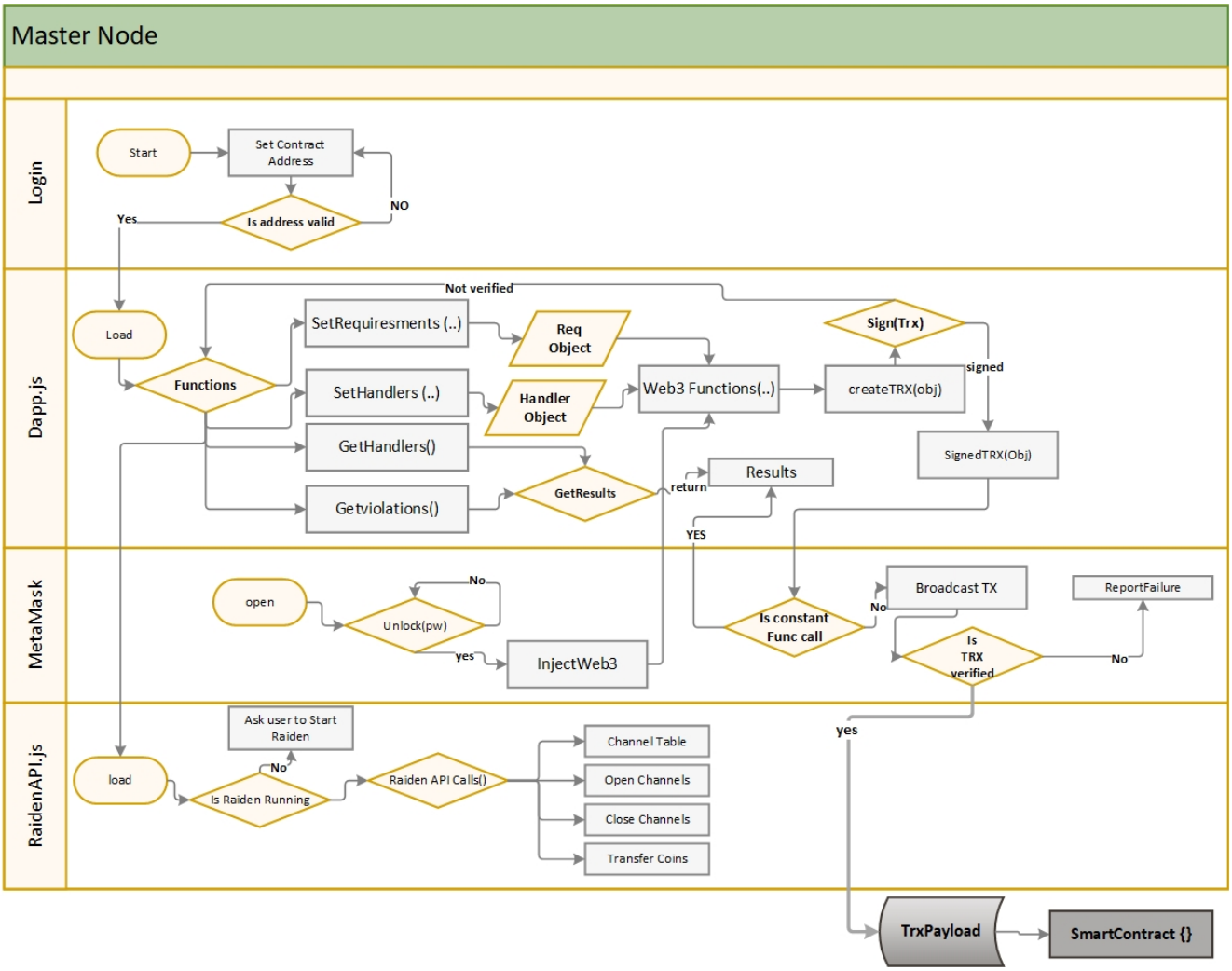

Figure 23: Work Flow diagram for Master Node

The System work flow of IoT Node is shown in figure 24. It is implemented as a command line program on RaspberryPI with the help of NodeJS. We have bundled all helper libraries including Web3.js, keyEthereum etc using node package manager utility. To start the shipping and monitoring process the supplier initializes the device with the tracking number. The device then contacts the SmartContract to read the shipping requirements associated with its tracking number. Transactions to Ethereum blockchain are relayed through remote Ethereum Nodes provided by Infura. It only provides API access to Ethereum nodes it does not store any keys nor does it handle transaction signing. The Sensors sub module calls functions in Utils modules to handle all Web3 calls. Utils module creates a web3 transaction to request a list of requirements the IoT Node need to adhere. This transaction is passed to Infura Ethereum Node which executes the request and returns results back to our node. Upon retrieving the shipping conditions from the Smart Contract the Sensors module initializes the required sensors i.e. temperature, light etc. Once the sensors have been initialized the IoT node enters the monitoring phase. Monitoring is further divided into two phases: Checking for violations or Monitoring, and logging (shipment data).

Monitoring: IoT Node checks sensor reading every few minutes to insure no violations have occurred. The monitoring period can be configured during initialization phase. If any violations are found, they are immediately sent to the Smart Contract to be recorded permanently. Violations are also added to the log. Recording violations in the blockchain is a state altering transaction, hence it must be signed by the Ethereum private key of the device. Utils module automates this process with the help of helper library ethereumjs-tx.js. Violations are signed with the Ethereum key of IoT Node and with the Post Quantum key of current shipper i.e. the one who breached the conditions. Violation packet sent to the blockchain is shown in figure 29. The PQ key changes when a new shipper takes control of the package or shipping procedure. Violation packets always carry the id of the current shipper along with the time and date of violations. This enables easy identification of the responsible party.

Logging: Logging basically creates a detailed record of package conditions in the form of a log. Each log entry contains the shipping data packet shown in figure 29. In short logging function records sensor values, package location, and shipper ID every 5 minutes. This log is written to IPFS as a mutable file object. Utils appends new log chunks at the end of previous log stored at IPFS. The generated hash is stored in the blockchain. This allows retrieval of complete logs after package has been delivered. These logs are useful for analytics purposes. The IPFS log interval can be configured during the initialization phase. Logging and Monitoring continue as long as the package is in transit, and the shipping status remains un delivered.

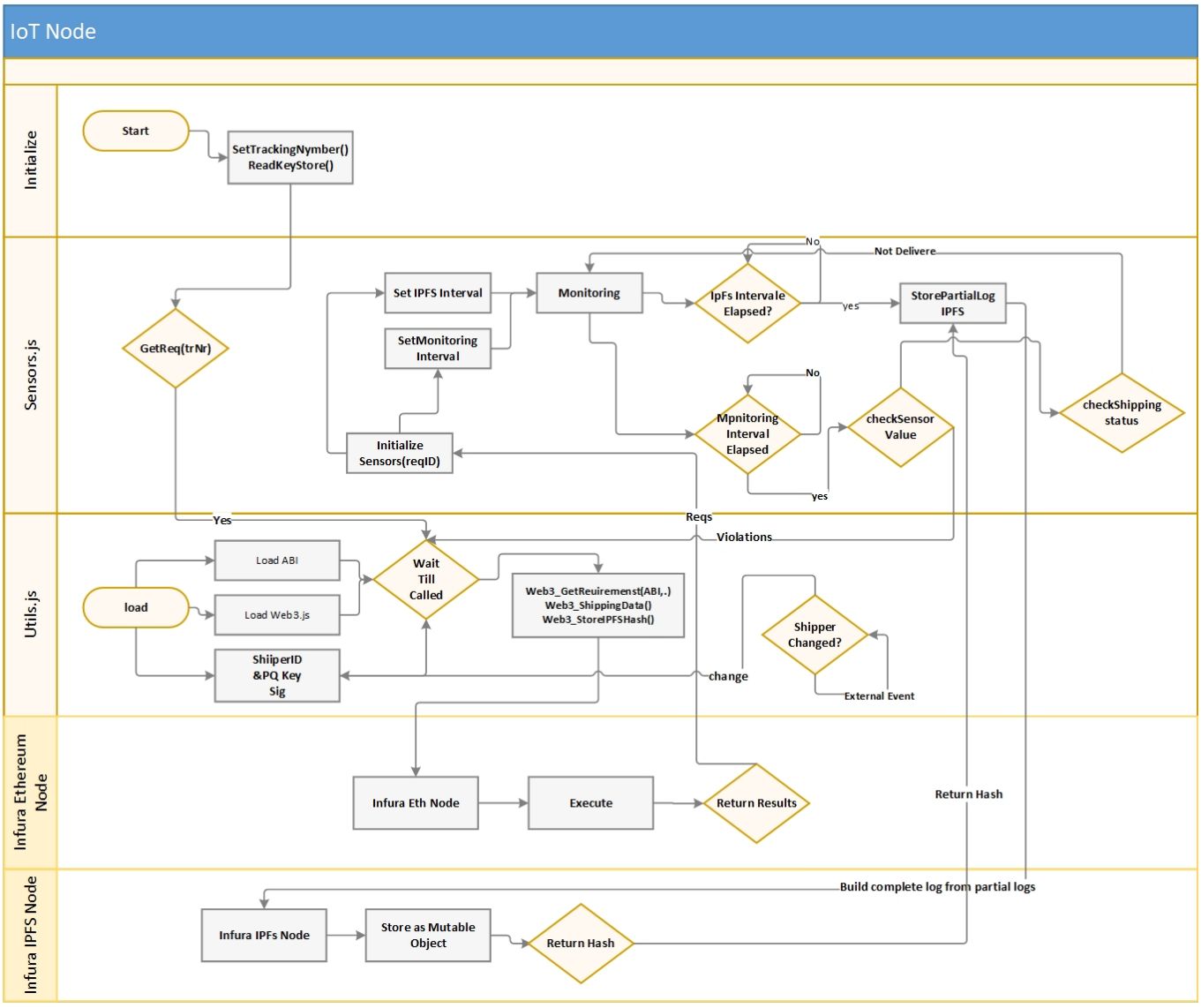

Figure 24: Work Flow diagram for IoT Node

This section presents important design elements of the Shipment Tracker Smart Contract. We have defined custom mapping and list structures to function as a logical database in our Smart Contract. This database saves tracking information, supplier access rights information, and shipping requirements. The primary key that logically binds different data structures is the tracking number. This tracking number is unique for each shipment or package. It is used as the key to store shipment data in the correct Solidity mapping. A mapping is a sort of hash table which is virtually initialized so that all possible keys exist in the table. In essence it is a key value pair mapping as shown in the figure 25. We have three main types of mappings in the Shipment Tracker contract. TrackingInfo mapping stores violations and tracking data associated with every package. This is described in 6.5.4. HandlerList mapping defines the rights of IoT Nodes and shippers (see 6.5.2). It tells which handler or Node is able to send violations and data to be stored in the TrackingInfo mapping. RequirementList mapping is responsible for defining requirements according to which each individual package must be handled. Our design relies on custom defined Solidity Modifiers that prevent shippers and IoT Nodes from modifying unauthorized sections of the database. Function modifiers are explained below:

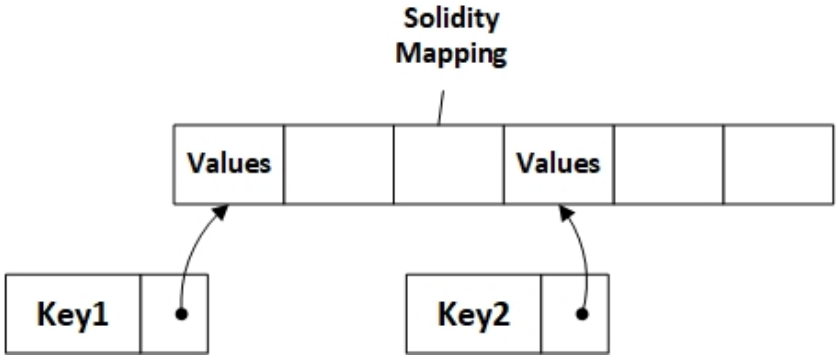

Figure 25: Sample Solidity Mapping

Function Modifiers

Modifiers are a convenient way to modify a functions behavior. They make sure that certain conditions are met before a contract function is executed. They are most commonly used to restrict access to certain parts of the program. When a function is called its modifiers are executed first to decide if the calling user or private key has the right to read or change the contract state. Shipment Tracker contract has three main modifiers as shown in figure 26. They are described in 6.5.1. Together these three modifiers perform authorization checks on different functions of the smart contract.

6.5.1 Shipment Tracker – Contract Work Flow

The functionality of the important sub modules of the Smart Contract are explained in the diagram presented in figure 26. The AuthorizeHandlerAddress function stores the IoT Node's Ethereum public key and Shippers ID and PQ keys. This data is stored as Handler Object in the HandlerList mapping. This mapping uses the tracking number as keys. Each key in the mapping corresponds to a dynamic list of handler objects as shown in figure 27. This function also insures that handler objects in all handler lists are unique. CheckAuthorization is the combination of two modifiers i.e. "onlyOwner" amd "onlyAuthorized". The first modifier insures that only the contract owner i.e. the person in possession of the private key that deployed the contract, can make certain types of changes. This modifier is used for defining requirements on individual packages as explained in section 6.5.4. The company is the only party that is able to define package requirements in the Smart Contract. "onlyAuthorized" modifier insures that shipping data and violations sent from IoT Nodes are saved in the correct section of the TrackingInfo mapping. It prevents unauthorized devices from modifying this mapping. The VerifySignature function shown in the diagram performs Post Quantum signature verification on the data received from the IoT Node. "OnlyIfSigMatch" modifier uses this function to insure that data and violations sent to the blockchain are correctly signed by the shippers PQ key. This insures data integrity. It insures that correct party could be held accountable for violations. This is particularly important in the cases where there is more than one shipper involved in the shipping procedure. To send violations to the blockchain IoT nodes call the LogTrackingInfo contract function. IoT Node sends current shippers ID in plain text form along with violations. Violations are encrypted and signed by the shippers PQ key. LogTrackingInfo uses the unencrypted shipper ID to locate shippers PQ key stored in the HandlerList mapping. It than proceeds to decrypt the encrypted information and calls the ChkViolation function. This function checks if the tracking data sent to contract is a violation according to the defined requirements. Once a requirement violation is verified it stores it in the TrackingInfo mapping and fires the violation event. Master Node and shippers can subscribe to violation events to get critical real time information related to package conditions. There are several Get Functions defined in the contract to read stored information related to a package. These include functions to read list of shippers authorized to handle the package, requirements according to which this package should be handled, and any and all violations incurred during shipping process. Finally, IoT node uses the SetIPFS function to store IPFS hash address of the detailed log created during shipping. This log can be retrieve from anywhere just by knowing the hash address.

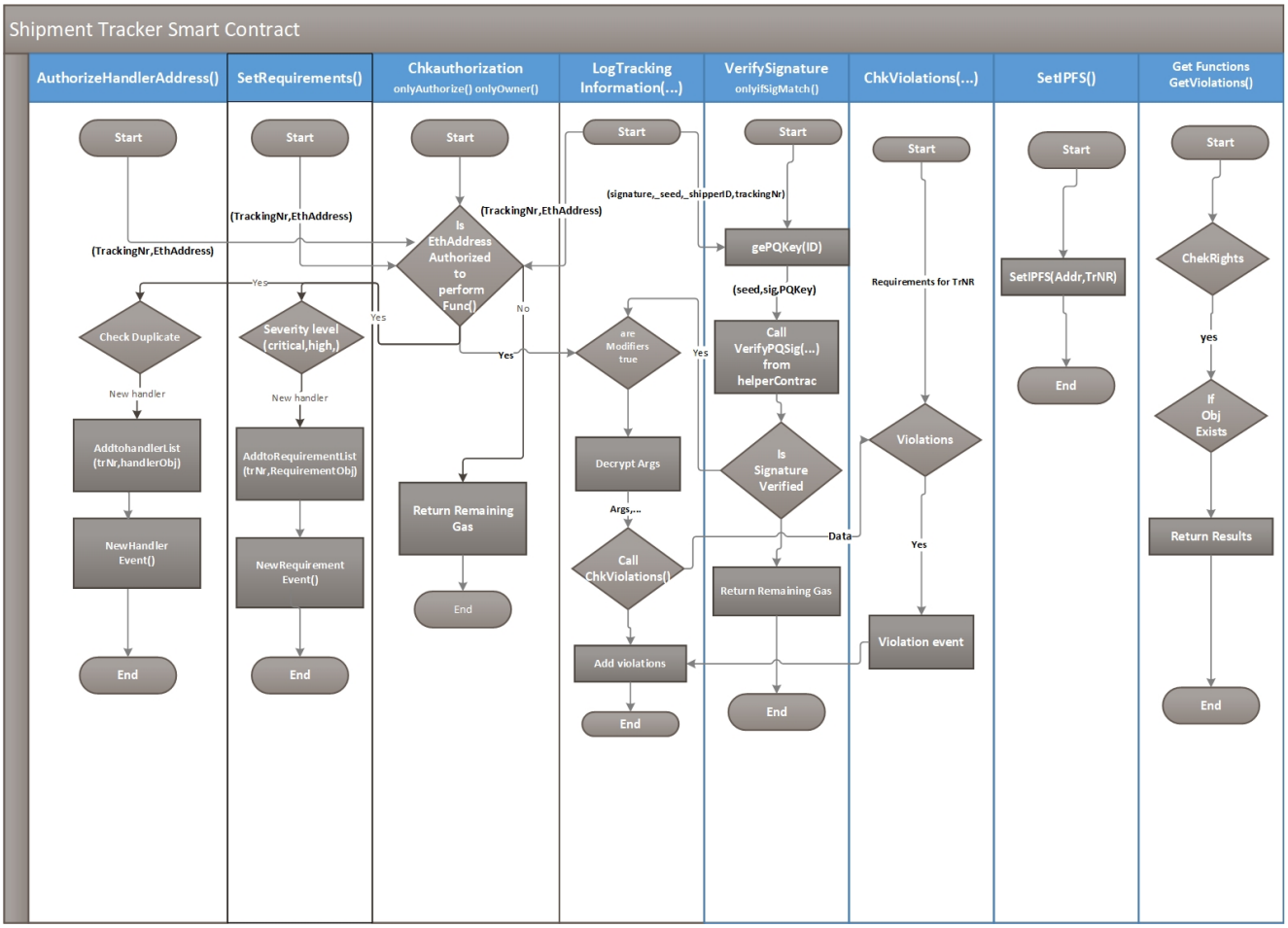

Figure 26: Shipment Tracker work flow

6.5.2 Handler Authorization

The AuthorizeHandlerAddress (see fig 26) allows Shippers and IoT Nodes to communicate shipping data and violations with the Smart Contract. Each package has one or more handlers or shippers who are authorized by the company to handle packages during shipping. The list of authorized handlers is stored in the HandlerList mapping shown in figure 27. This mapping uses the tracking number of the package to create a dynamic list of handler objects. Each handler object represents only one shipper. Each object contains the shipper ID, their PQ key and the IoT Node Eth address. The IoT nodes Eth address is used for communicating shipping data with the blockchain. The company adds shippers or handlers to the HandlerList mapping by generating a transaction using the Master Node's graphical interface. This is a state altering operation; hence it must be signed using company's private Ethereum key. Contract design dictates that this key must be the same as the one that was used to deploy the Shipment Tracker Contract. This key is represented as Master Node Eth Address in the figure 27.

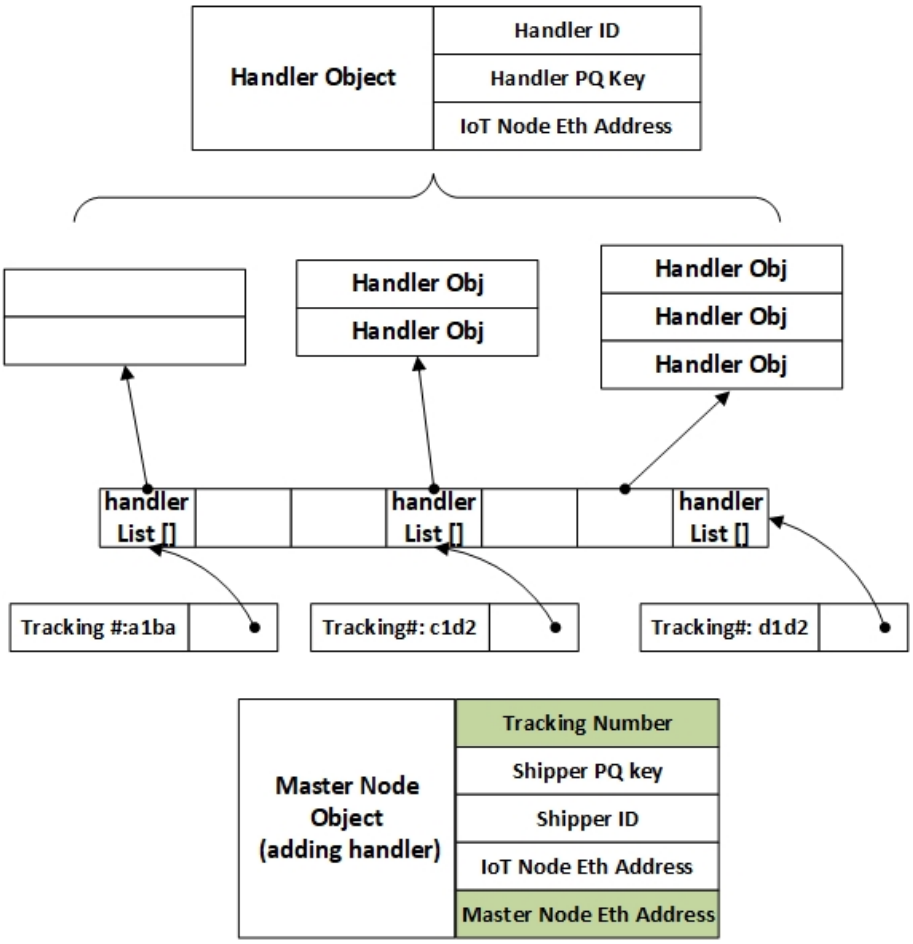

Figure 27: Adding a new handler to the Handler List

Revoking Authorization

If for any reason the company wishes to revoke the authorization of a shipper. They can do so by calling RevokeAuthorization function using the Master Node. This function removes the handler object associated with the shipper in the handler list. This is also a state changing operation and as such must be completed using a signed transaction.

6.5.3 Requirements

Setting Shipping Requirements is similar to authorizing handlers as evident from the work flow diagram in figure 26. Adding or removing requirements is a state changing operation and as such can only be executed using signed transactions. A miner verifies that the transaction is correctly signed by company's private key. It then passes the transaction payload to the Smart Contract. The transaction payload in this case is a call to SetRequirements function of the Shipment Tracker Contract. The contract verifies weather the Ethereum address that made this call has the right to execute this function. This is achieved using the "onlyOwner" function modifier. The SetRequirements function parses the received Master Node object shown in figure 28 and extracts the tracking number. This tracking number is the key for storing

requirements in the RequirementsList Mapping. The function then creates the Requirement object. This object is saved in the dynamic array that corresponds to the tracking number in the mapping as shown in figure 28. The requirement object can hold different types of requirements e.g temperature, light, pressure, Humidity etc. This necessitates that the requirement object is as generic as possible. This object has two essential properties i.e. requirement ID and Severity Level. Everything else is optional. Requirement ID uniquely identifies a type of requirement e.g. temp for monitoring package temperature. Severity level can have one of four values e.g. Critical, High, Medium and Low. Max Threshold and Min Threshold properties of the requirement object define the maximum and minimum values that must never be exceeded. As an example consider a medical substance that must be transported under strict climate controlled conditions. The package temperature must never exceed 25°C (Max Threshold), neither should it fall below -10°C (Min Threshold). there could be conditions that only require one type of threshold i.e. either max or min. The two requirement flags i.e. Min Flag and Max Flag enable the IoT node to determine, which threshold is relevant for any given requirement ID. If both flags are true than then it checks the sensor values against both thresholds.

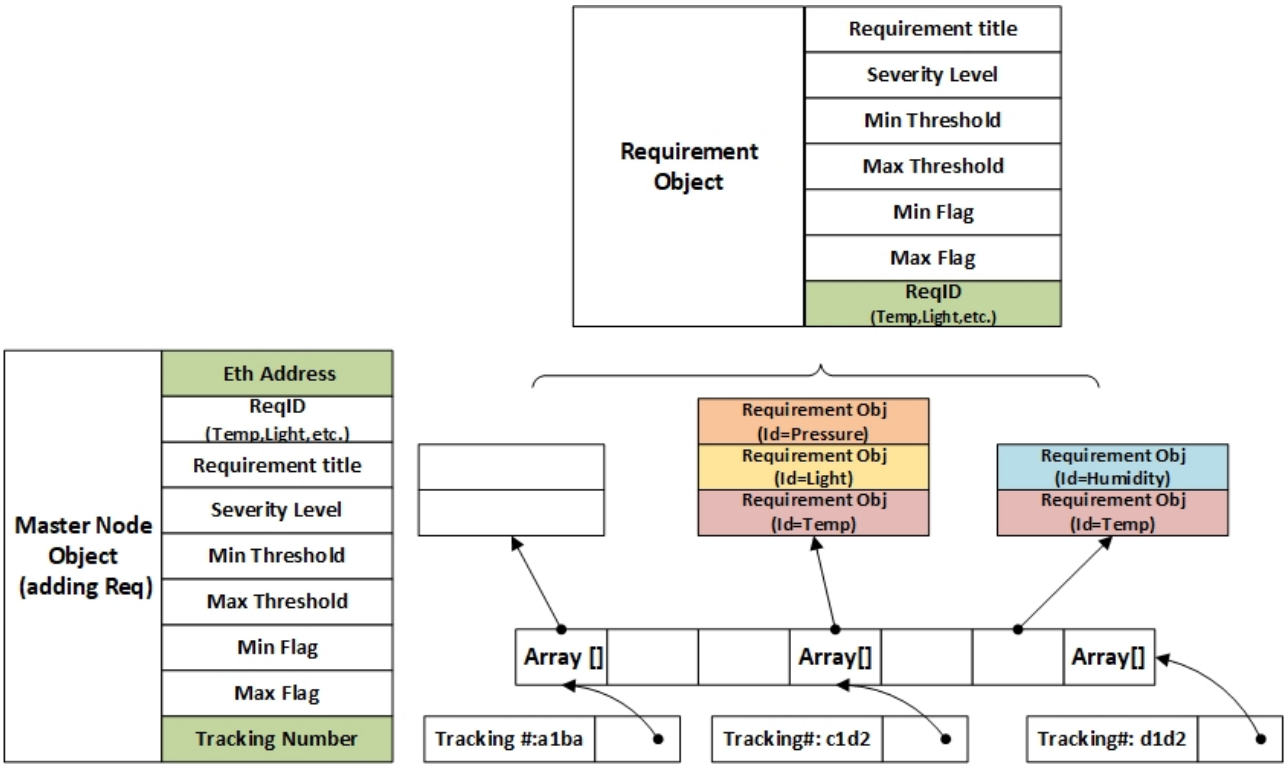

Figure 28: Setting shipping requirements for a package

IoT Nodes and Requirements: IoT Node requires a set of requirements to begin monitoring package conditions. It receives these requirements by sending its tracking number in the form of a get request to the Shipment Tracker contract. The contract responds to IoT node's request by sending all the requirements associated with a particular tracking number. This requirement ID determines which environmental conditions the IoT Node will monitor. These nodes have been tested with temperature,

humidity, pressure, and GPS sensors. Additional sensors can be easily integrated without requiring any major changes to the design.

6.5.4 Violations

IoT nodes are responsible for monitoring shipping conditions according the defined requirements during shipping. Each shipping cycle is logically separate from others. It could be that an organization has different set of suppliers for different products. IoT nodes and suppliers only have access to the parts of the database (mappings) that are associated with their package tracking number as shown in figure 29. This insures that shipment data and violations remain logically separated even with multiple parallel supply lines. The shipping data object sent from the IoT Node has the tracking number, which is used as the key in the Solidity mapping as shown in figure 29. This object is signed with the post quantum key of the shipper. This insures data integrity even in the presence of Post Quantum threats. IoT Node can be optionally configured to encrypt certain parts of the shipping data object to protect confidentiality during transit. Shipment Tracker contract parses the received shipping data object and creates the violation object shown in figure 29. This object is passed to the check violations functions and to confirm shipping violations and to fire appropriate violation events. Company and suppliers can subscribe to these events and respond in real time. All violations are permanently stored on the blockchain.

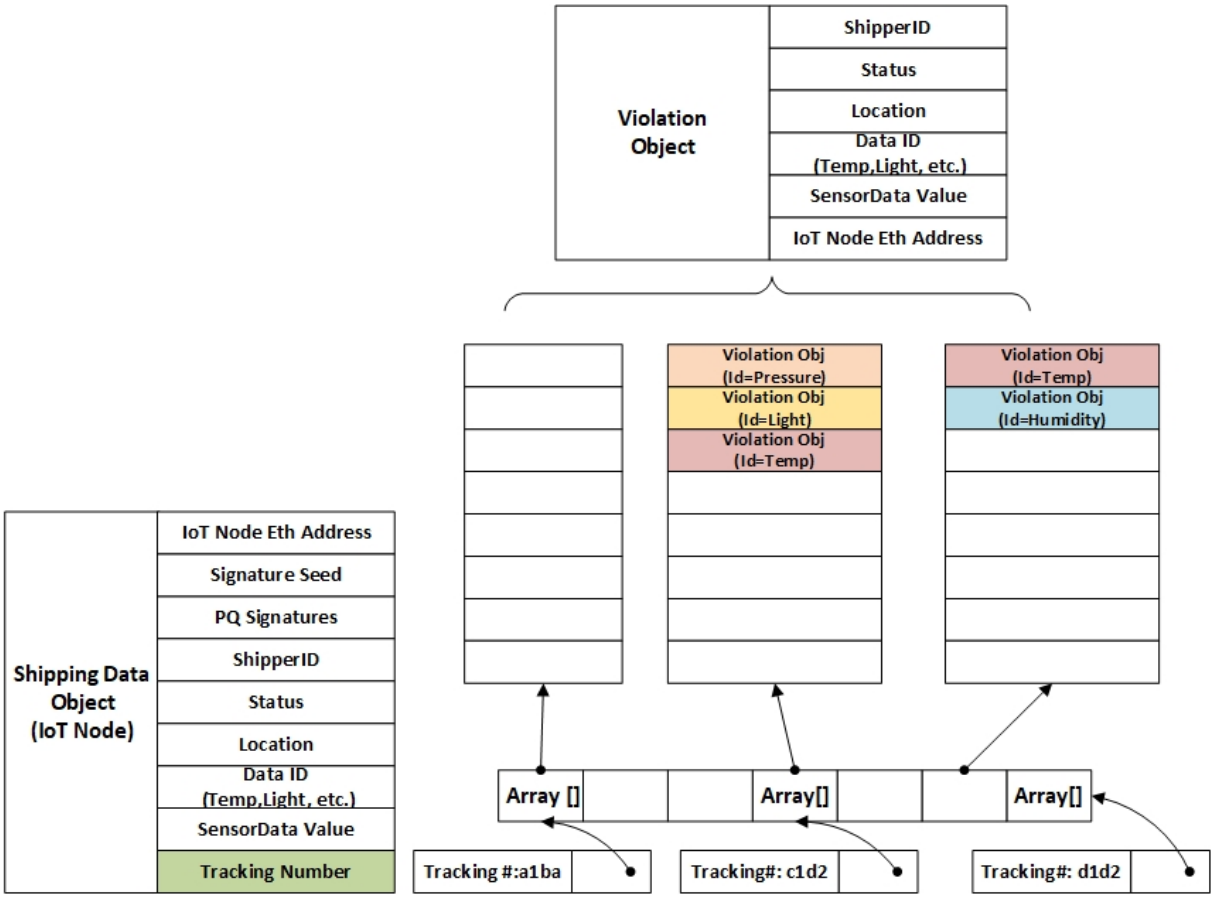

Figure 29: Adding shipping data and violations to the blockchain

6.6 Securing the System against Post Quantum Adversaries

The biggest advantage of a blockchain is the security benefits afforded to them due to their decentralized nature. Blockchain security model relies on a combination of cryptographic primitives, secure hashing algorithms, digital signatures and distributed nodes. It is almost impossible for a single attacker to hack the blockchain. Relying on traditional means any successful attacker would need to control more than half the nodes or network hashing power to create fraudulent transactions. This is known as a 51 percent attack. This attack is almost impossible in large blockchain networks such as Bitcoin and Ethereum. However, one thing that could pose a threat to blockchain are Quantum Computers. They are a promising area of research, but could one day prove to be an existential threat to the blockchain. This is because they can at least in theory break the cryptographic primitives that blockchains rely on. One of our goals is to design a system with architecture primitives that could defend against any future quantum adversaries. It needs to prevent attackers from modifying critical parts of the system such as violation data. This system can be explained with the help of algorithm 1.

Our proposed security model relies on a combination of rights management based on Solidity modifiers and post quantum algorithms for generating secure digital signatures. Important parts of this solution are explained in detail in section 6.5. The post quantum signature scheme for this scenario uses lattice based algorithms developed at TU Darmstadt such as [Joh11]. Alternatively, we can also use schemes recently submitted to NIST. The PQ algorithm is implemented in a separate library contract as shown in figure 20. Our Shipment Tracker contract uses a wrapper function to call the signature verification function of the helper contract. This enables us to easily change the post quantum algorithm without requiring any major changes in the Shipment Tracker Contract. This security model only protects data integrity, additionally it can be used to protect confidentiality during transport. We can not guarantee confidentiality of data once it has been stored on the blockchain. This is because everything stored on the blockchain is by default visible to every node having a copy of the ledger. A malicious node can dig through the stored blocks and get PQ keys of authorized shippers using eth.getStorageAt function. They can then use these keys to decrypt data.

```
Initialize the IoT device;
Data: Get Sensor Data
while Shipping status not Delivered do
    Check violations;
    if violation detected then
        Start signature procedures;
        Seed = random(256);
        PQSign = Sign(Message,SecretKey);
        message = (dataID || location || status ||TimeStamp);
        Encrypt function arguments;
        TrxPayload = Functioncall(encryptedArg1,encryptedArg2,...,PQsign,TimeStamp,shipperID);
        CreateSignedEthTransaction(EthpvtKey,TrxPayload);
        BroadcastTransaction;
        save violation and shipping data in IPFS;
    else
        save shipping data in IPFS;
    end
end
Blockchain verifies GAS and Ethereum Signatures;
while Miners available do
    checkAuthorization(IoTNode public key);
    get shipper PQ key from Shipper ID;
    Smart contract verifies PQ signatures;
    if Verify(PQkey,TimeStamp,signature) then
        if (Eth Signatures Authorized and PQ Signatures Authorized) then
            message = DecryptArguments(cArg1,cArg2,...);
            execute the function;
            fire Events;
        else
            Return Error;
        end
    else
        Return Error;
    end
end
```

Algorithm 1: System Security Model Algorithm

6.7 Implementation Challenges

6.7.1 Problems with Raiden and Fixes

We faced most difficulties while getting Raiden client to work reliably. When we started developing our solution Raiden was in its developer preview / beta stage. We choose it because at that time it was the most mature off chain transaction solution available for the Ethereum Echo system. The other solutions under consideration was Perun, which at the time was still working on a proof of concept implementation. Raiden was particularly difficult to setup and get off the ground. To get started it first needs to completely sync with the Ethereum blockchain. This process can take up to 3 to 5 days to complete in the worst case scenario. Even after syncing it required that the machine hosting Raiden remained on to as continue syncing with new transactions. If we turned off the machine, we would have to wait for it to catch with the blockchain before we could use it again. Once it was setup we faced further problems making direct payment transfers. The direct payments worked intermittently. This problem led us to contact the Raiden team to find a solution. We worked closely with the development team to debug the issue. It was concluded that the Raiden Nat punching module was the culprit. This module is required to punch through firewall and local Nat routers. This module did not always work reliably. In order to make direct payment transfer Raiden nodes need to be able to ping each other. In the end we found a work around which worked 90 percent of the time. At the time of last contact Raiden developers were still working on a permanent fix for the Raiden Nat punching module. Raiden has now also updated its official documentation [Rai17] to add a set of best practices to be implemented by the user. These best practices are closely aligned with the work around that we discovered to resolve some of the situations we were facing.

6.7.2 Problems with Web3

The Web3 is a JavaScript helper library which allows us to communicate with Ethereum nodes over HTTP connections. In our project we use node package manage NPM to manage dependencies for us. NPM always installs the latest official version of the required libraries. When we started development the latest version was Web3 0.20.6. Half way through development a new version of the Web3 library i.e. 1.0.0 was released. The new update radically overhauled how the library was implemented and worked. It changed functional interfaces and how they are called. The new update also changed the manner in which the functions could be used by making major architectural changes in Web3 function calls. This forced us to re-implement and overhaul application code for Master Node and IoT Node. In version 0.20.6 a Web3 function calls behaved synchronously i.e. execution of the module would halt until the call was resolved. This meant we could count on the next function to always get correct results from the previous function calls. In the new 1.0 update Web3 function calls behave asynchronously i.e. it does not halt any part of the program. This means that after a transaction call was made it would return immediately and return control to the next function inline even though the previous function has not resolved yet. Instead of returning values new update returns JavaScript Promises. This means we

are required to use callbacks to resolve promises correctly. We also had to re design how we passed values from individual sub modules to other modules. In version 0.20.6 we passed the values directly to other modules, in version 1.0.0 we had to redesign IoT Node sub modules to use Async/Await for communicating values across module boundaries.

7 Testing and Results

7.1 Testing Strategy

This Chapter details testing methodologies used to verify functional and non-functional aspects of the Master Node, IoT Node, and the Shipment Tracker Contract.

7.1.1 Shipment Tracker Contract

Every function of Shipment Tracker contract was tested extensively using the Truffle framework and Remix Ethereum IDE. Truffle is a framework for testing and deploying Smart Contracts. Truffle can be installed using Node Package Manage (NPM). It provides an Ethereum Virtual Machine TestRPC client known as Ganache CLI. This mimics the Ethereum network in a sandboxed environment. This client runs on the local machine and simulates the blockchain Network. The test RPC client is useful for testing Smart Contracts and Decentralized applications before deploying them on the actual blockchain. It makes the process of testing Ethereum Smart Contracts very simple. Unlike Ethereum MainNet or TestNet we do not have to wait for transactions to be mined and confirmed. Ganache CLI is a self-contained private blockchain network on a machine. Every transaction sent to this blockchain is immediately mined and confirmed. This allows us to quickly test small changes in the contract.

7.1.2 Testing Complete System Interaction

Our complete system cannot be tested using the Truffle framework. As stated earlier the TestRPC provided by Truffle is a sandboxed blockchain, private to the system where it is running. In addition, the blockchain ledger is wiped cleaned on closing the TestRPC client. To test complete system interactions i.e. Smart Contract, IoT Node, and Master Node the contract was deployed on Ropsten Testnet. Ropsten Test Net closely mimics the main Ethereum blockcahin in terms of its operations. However, unlike the main blockchain Ropsten Eth can be requested for free from Ropsten faucet (https://faucet.ropsten.be/). This enables any one to test complete systems without spending money.

7.1.3 Master Node

Master Node has several parts including the GUI, Smart Contract interface module and Raiden Network Interface. The GUI and Raiden Network were tested manually. In order to test Master Node interaction with Smart Contract two types of testing was performed. TestRPC was used to test individual functions against locally deployed version of the Shipment Tracker Contact. Secondly, to test full system interaction Master Node was tested using Shipment Tracker deployed on the Ropsten test network.

7.1.4 IoT Node

We used Ropsten testnetwork to test the IoT Node. Validation testing of Individual sub modules was performed using Truffle TestRPC. Each major update would go through a suite of validation testing before being deployed on the TestRPC. A SensorTest.js class was deployed on the development machines which provided simulated sensor values for development testing.

7.2 Results

7.2.1 Module Testing?

unit testing was mostly done on ganache which is self contained and isolated Ethereum virtual machine program. This replicates ethereum network in a sandboxed envirionment as each ethereum node on the network runs its own implementation of the ethereum virtual machine.

-testing storing minute by minute logs on the blockcahin i.e. to justify how we came across the case that its not feasible to store everything on the ethereum blockchain

7.2.2 Changing GPS coordinates

In order to simulate channgin GPS locations a rotary switch was used which would send preconfigured coorodinate to the IoT node function listening for GPS coordinates. May be this should be moved to testing environment section ?

7.2.3 Testing setting Requirements

mention how setting requirement transaction works, no restrictions on the requirement ID except they should be unique. IoT node receives the requirements and checks if it has a sensor able to monitor the requirement ID. If no such sensor is found it starts the monitoring procedure with for thise requirements that it has sensors for. show IoT node getting requirement screen shot.

7.2.4 Scenario - I

Testing with one shipper - no violations Testing with multiple shipper - no violations, rotary switch acts as shipper change indicator

7.2.5 Scenario - II

Testing with one shipper - one violation testing with multiple shippers - one violations i.e. see if the violating shipper is easily traceble in the end

7.2.6 Scenario - III

Testing with one shipper multiple violations. Testing with multiple shippers - multiple violations. see if each shipper is only hit with their particular violations.

7.2.7 Testing Token Transfers and Channels deployement - Raiden

Put screen shots for token transfers if found, Mention how token transfer is immedeiate, mention synching time with the blockchain. give the list of best practices and other things you found helped establishing channels

7.2.8 IPFS

?

7.3 Evaluation

The evaluations presented in the next few sections are based on approximations using data like Eth price, and Gas Price that is extremely volatile or difficult to determine.

Tools Used: etherscan.io, ethgasstation.info

7.3.1 Transaction Speed and Cost

Section 4.1.4 showed that the total transaction cost can be calculated by multiplying, the gas used by transaction, with gas price. The user is allowed to set their own gas price. This price usually determines how fast on average the transaction will be confirmed. Miners usually prefer transactions with higher gas prices over others. If the price is too low miners will simply ignore the transaction. Minimum safe gas price can be found using ethgasstation.info. The gas price depends upon network conditions. The table 2 given below shows gas prices on two different dates. The total gas required by a transaction depends on the type of operation it will perform and will largely remain same for the same type of transactions.

7.3.2 Contract Deployment Cost

Initially we intended to use separate Smart Contracts for individual shipping cycles. The idea was a new contract would be deployed at the start of each shipping cycle. We quickly realized this would generate lots of segregated contracts to keep track of for a large organization. It would also be expensive in terms of operations, deployment and maintenance specially on a public blockchain like Ethereum. Figure 30 shows a transaction deploying Shipment Tracker Contract using MetaMask. Table 3 shows cost of

Date	Speed	Gas Price (gwei)
16 September 2018	SafeLow (<30m)	3
	Standard (<5m)	5
	Fast (<2m)	6
December 4 2017	SafeLow(<30m)	25
	Standard(<5m)	45
	Fast(<2m)	57

Table 2: Gas Price and transaction times

deploying Shipment Tracker and Helper contracts based on different Eth prices. The prices in this table are given after converting from Eth to US dollars. We have used two different Eth prices to elaborate the worst case and average case scenarios. Ethereum prices can fluctuate by significant amounts from one day to another. Deploying a contract can be the most expensive transaction in any Ethereum based system. Considering this and other factors listed above we chose to design a contract that could handle multiple shipping cycles in parallel. This contract is more complex and hence costs more to deploy compared to the contracts that would handle only one shipping cycle. However, this contract has to be deployed once and hence is much more efficient in terms of operations and maintenance.

Contract	Gas Limit	Deployment Cost	Gas Price	Price of 1 ETH
Helper.sol	922416 gwei	10$	15 gwei	222$
ShipmentTracker.sol	6468201 gwei	70.2$		
Helper.sol	922416 gwei	116.4$	55 gwei	1200$
ShipmentTracker.sol	6468201 gwei	388.092$		

Table 3: Cost of contract deployment

7.3.3 Cost of Storage on the Blockchain

In 4.4 it was mentioned that it becomes prohibitively expensive to store large chunks of data on the blockchain. Table 4 shows the cost of storage on the Ethereum blockchain. This table takes the best case scenario into consideration. In this scenario the minimum observed Eth price was used for calculation. We are only interested in logging values from one sensor, and additionally the log interval is quite large I.e. once every 1 hour. In reality we want our system to have in depth logs of each and every activity and the logging interval should be as small as possible. During IPFS testing we kept the logging interval at 5 minutes. This table makes it obvious it is not feasible to store logging information in the blockchain.

7.3.4 Cost of storing Violations and Requirements

Sending violations or Requirements to the Shipment Tracker contract takes almost equivalent amount of gas. For the calculations shown in table 4 we used a transaction generated from the Master Node. The transaction was used to set temperature requirements for the IoT node. The gas price for this transaction was kept at 20 gwei. This transaction spent 0.00484134 Eth worth of gas. Which is equivalent to 1

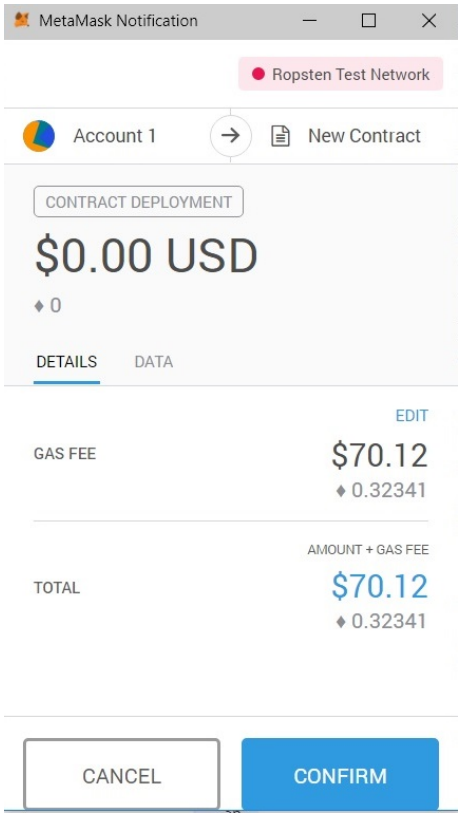

Figure 30: Deploploying Shipment Tracker contract using MetaMask

Data size	Gas Required	Transaction Cost (Eth)	ETH Price	Interval	Sensors	Daily Cost (Eth*Eth Price*24)
256 bit word	20000	0.0006	222$	1 hr	1	3.1968$
1MB (31250 256-bit words)	625000000	18.75				99,900$
10MB	6250000000	187.5				999,0000$
100MB	62500000000	1875				9,990,000$

Table 4: Cost of Storage on Ethereum Blockchain

dollar according to the Eth pricing as of 17 August 2018. Alternatively, if we take into account Eth pricing from January, the cost of this transaction becomes 5 dollars. Our tests have shown that sending violations to the blockchain takes almost the same amounts of gas. The table 4 given below shows daily operational cost based on best case and worst case scenarios. The worst case simulates violations issued for all four sensors attached to the IoT Node. In this case we are only storing each type of violation once, repeated violations will be stored in IPFS. The worst case also simulates a large organization with multiple suppliers and multiple shipments on route at the same time. The table calculations are based on 5 parallel supply lines.

	Number of Violations	Transaction Cost	Gas Price	Price of 1 ETH	Number of Supply lines	Daily Cost
Best Case	0	0$	20 gwei	222$	1	0$
Worst Case	4	1$			>5	20$
Best Case	0	0$	20 gwei	1200$	1	0
Worst Case	4	4$			>5	80$

Table 5: Daily Operational Cost

points to elabortae Make the design flexible i.e. so that it does not require always on internet connection. i.e. it communicates shipping and logistical data whenever internet or network connection is available. Use Perun to get the entire smart contract offchain to reduce dependency on on chain and resiliance against network congestions. etct etch
log files uploaded to IPFS can be encrypted with either public key or secret key techniques to provide complete data security further try and solve the confidentiality issue.? To conclude. . . Conclusion

In this thesis we developed a proof of concept system , This demonstrated the usefulness of such a system. A corporate or enterprise level solution will need additional supporting technologies and systems for it function smoothly. One area of improvement would be integrating RF chips in the IoT nodes that shippers Scan using their phones or RF readers to send change in shipper notifications etc. We could develop a smart phone application which could be used to scan packages and get all relevant information using NFC or Bluetooh. Managers and Supply chain engineers can use this application to subscribe to important events such as change in shipper or custom checks etc. to get real time information every where. This could help them plan better supply chain processes in the future.

Our system can reduce cost, increase efficency, and increase transparency. It reduces cost by increasing automation, data is sent autmatically from IoT nodes to the bloccckhain, The company and shippers does not need to hire dedicated persons for entering data in suplly chain monitoring systems which is the case in traditional systems. Shippers and Company reps can subscribe to important events.

Efficency is increased, by eliminated human errors, and providing real time critical data to the relevant stake holders.

This system establishes trust between parties by giving every stake holder access to the same information. Shippers, suppliers and company execs all have access to the same data, and the system is not under the control of any one party.

8.1 Benefits

I.e allows complete transparency to all participants of the supply chain. Enables / Gives end users or customers to have complete confidence that the product was stored, shipped and handled in accordance to strict safety standards and regulations. The transparency brought by this solution to the supply chain life cycle makes the job of government regulators and safety inspectors much simpler.

Although our test case calls for only monitoring environmental conditions and location of a shipment or package, This technology can be easily extended to monitor other critical events like custom checks or quality checks etc. This has the potential to reduce delays at borders, custom procedures could be automated with IoT node securely communicating custom declarations to the systems at the border.

Similarly quality and compliance checks can be expedited, IoT package would send quality and comliance information to the concerned parties in advance or on demand. We can greatly reduce the number of humans involved in these processes by automating processes like custom declarations and compliance certifications.

Improvements before shipping starts the raspberry pi must be configured with the correct tracking number. The supplier must do this manually, how ever this can be automates easily by giving a static device number to each sensor node and then assigning new tracking number to a device each time raspbery pi / iot node starts. Wehn an IoT node starts it will query the remote server with its deviceID to get a tracking number for the new shipping cycle.

We can integrate a supplier rating system in our dapp to keep track of ratings. The supplier and shipper would be given a bad rating for less svere violations. Each violation carries a severity level i.e. medium to low would result in a bad rating. Organization can use the ratings to keep track of the quality of service provided by the shipper or supplier. Mention another benefit of using a private blockchain would be to

List of Figures

List of Tables

Bibliography

[Ada14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. https://blockstream.com/sidechains.pdf, 2014. [Online; accessed August 25, 2018].

[Ahm16] Ahmed Banafa. A Secure Model of IoT with Blockchain. https://www.bbvaopenmind.com/en/a-secure-model-of-iot-with-blockchain/, 2016. [Online; accessed August 26, 2018].

[Air18] Airbus. Airbus supply chain. https://www.adsgroup.org.uk/wp-content/uploads/sites/21/2016/02/Integrated-Supply-Chains.png, 2018. [Online; accessed August 21, 2018].

[And14] Andreas M. Antonopoulos. Mastering Bitcoin. https://github.com/bitcoinbook/bitcoinbook, 2014. [Online; accessed August 23, 2018].

[Ben14] Juan Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

[Ben15] Benjamin Herzberg. Blockchain: the solution for transparency in product supply chains. https://www.provenance.org/whitepaper, 2015. [Online; accessed August 25, 2018].

[Ben16] Ben Dickson. Decentralizing IoT networks through blockchain. https://techcrunch.com/2016/06/28/decentralizing-iot-networks-through-blockchain/, 2016. [Online; accessed August 26, 2018].

[Ber18] Bernard Marr. Here Are 10 Industries Blockchain Is Likely To Disrupt. https://www.forbes.com/sites/bernardmarr/2018/07/16/here-are-10-industries-blockchain-is-likely-to-disrupt/#20d06348b5a2, 2018. [Online; accessed August 25, 2018].

[Bit12] Bitcoin wiki contributors. Proof of stake — Bitcoin wiki. https://en.bitcoin.it/wiki/Proof_of_Stake, 2012. [Online; accessed 25-August-2018].

[Blo16a] Blockchain Hub. Blockchains and Distributed Ledger Technologies. https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/, 2016. [Online; accessed August 25, 2018].

[Blo16b] Blockgeeks. Smart Contracts: The Blockchain Technology That Will Replace Lawyers. https://blockgeeks.com/guides/smart-contracts/, 2016. [Online; accessed August 31, 2018].

[Blo18] Blockchain.info. Median confirmation time. https://www.blockchain.com/charts/median-confirmation-time?timespan=all, 2018.

[BM18] Rachid El Bansarkhani and Rafael Misoczki. G-merkle: A hash-based group signature scheme from standard assumptions. Cryptology ePrint Archive, Report 2018/274, 2018. https://eprint.iacr.org/2018/274.

[CDE+12] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.

[DEFM17] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. Cryptology ePrint Archive, Report 2017/635, 2017. https://eprint.iacr.org/2017/635.

[DFH18] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. Cryptology ePrint Archive, Report 2018/320, 2018. https://eprint.iacr.org/2018/320.

[Eth17a] Ethereum Wiki Contributors. Ethereum sharding research compendium. https://github.com/ethereum/wiki/wiki/Sharding-introduction-R&D-compendium, 2017. [Online; accessed 25-August-2018].

[Eth17b] Ethereum wiki Contributors. Patricia tree — Ethereum wiki. https://github.com/ethereum/wiki/wiki/Patricia-Tree, 2017. [Online; accessed 31-August-2018].

[Eth18] Ethereum wiki Contributors. Proof of stake faqs — Ethereum wiki. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs, 2018. [Online; accessed 25-August-2018].

[Fil16a] Filament Foundation. Distributed Exchange and the Internet of Things. https://filament.com/assets/downloads/Filament%20Foundations.pdf, 2016. [Online; accessed August 26, 2018].

[Fil16b] Filament Foundation. Distributed Exchange and the Internet of Things. https://filament.com/assets/downloads/Filament%20Security.pdf, 2016. [Online; accessed August 26, 2018].

[Fre17] Fred Ehrsam. Scaling Ethereum to Billions of Users. https://medium.com/@FEhrsam/scaling-ethereum-to-billions-of-users-f37d9f487db1, 2017. [Online; accessed August 25, 2018].

[GoC17] GoChainGo. Why Scaling Public Blockchains Is a Lot Harder than just Increasing Block Size. https://medium.com/gochain/why-scaling-public-blockchains-is-a-lot-harder-than-just-increasing-block-size-5c5e7 2017. [Online; accessed August 25, 2018].

[Hud17] Hudson Jameson. Accounts, Transactions, Gas, and Block Gas Limits in Ethereum. https://hudsonjameson.com/2017-06-27-accounts-transactions-gas-ethereum/, 2017. [Online; accessed August 31, 2018].

[IBM16] IBM IoT blog. Apply IoT and blockchain for accountability and security. https://www.ibm.com/internet-of-things/spotlight/blockchain, 2016. [Online; accessed August 26, 2018].

[IBM17] IBMBlockchain. Securing the diamond trade with blockchain. https://www.youtube.com/watch?v=hHkc-DH0ep4, 2017. [Online; accessed August 25, 2018].

[Jam18] Jamie Redman. Miami Bitcoin Conference Stops Accepting Bitcoin Due to Fees and Congestion. https://news.bitcoin.com/miami-bitcoin-conference-stops-accepting-bitcoin-due-to-fees-and-congestion/, 2018. [Online; accessed August 25, 2018].

[Jef18] Jeff Coleman,Liam Horne,Li Xuanji. Counterfactual: Generalized State Channels. https://l4.ventures/papers/statechannels.pdf, 2018. [Online; accessed August 25, 2018].

[Jim18] Jimi S. Blockchain: how mining works and transactions are processed on the blockchain in seven steps. https://medium.com/coinmonks/how-a-miner-adds-transactions-to-the-blockchain-in-seven-steps-856053271476, 2018. [Online; accessed August 23, 2018].

[Joh11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss - a practical forward secure signature scheme based on minimal security assumptions. Cryptology ePrint Archive, Report 2011/484, 2011. https://eprint.iacr.org/2011/484.

[JP 15] JP Buntinx. An overview of the ethereum virtual machine. https://nulltx.com/what-is-the-ethereum-virtual-machine/, 2015. [Online; accessed 25-August-2018].

[JP16] Thaddeus Dryja Joseph Poon. The bitcoin lightning network: Scalable of-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2016.

[Kar18] Karim Sultan,Umar Ruhi,Rubina Lakhani. CONCEPTUALIZING BLOCKCHAINS:CHARACTERISTICS & APPLICATIONS. https://arxiv.org/pdf/1806.03693, 2018. [Online; accessed August 25, 2018].

[Lig15] Lightning Network. Lightning Network Scalable, Instant Bitcoin/Blockchain Transactions. https://lightning.network/, 2015. [Online; accessed August 25, 2018].

[Lys02] Anna A. Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, Cambridge, MA, USA, 2002. AAI0804606.

[Max18] Max Thake. What is Proof of Stake? https://medium.com/nakamo-to/what-is-proof-of-stake-pos-479a04581f3a, 2018. [Online; accessed August 25, 2018].

[mis17] How Blockchain Tech Is Changing Cloud Storage. https://www.belugacdn.com/blog/162663814938-how-blockchain-tech-is-changing-cloud-storage, 2017. [Online; accessed August 25, 2018].

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[Nat18] Nathan Reiff. Forget Bitcoin: Blockchain is the Future. https://www.investopedia.com/tech/forget-bitcoin-blockchain-future/, 2018. [Online; accessed August 25, 2018].

[Pat12] Patrick Kingsley. Financial crisis: timeline. https://www.theguardian.com/business/2012/aug/07/credit-crunch-boom-bust-timeline, 2012. [Online; accessed August 26, 2018].

[Pet15] Peter Frøystad, Jarle Holm. Blockchain: Powering the Internet of Value. https://www.evry.com/globalassets/insight/bank2020/bank-2020---blockchain-powering-the-internet-of-value---whitepaper.pdf, 2015. [Online; accessed August 31, 2018].

[Pro17] Protocol Labs. Filecoin: A Decentralized Storage Network. https://filecoin.io/filecoin.pdf, 2017. [Online; accessed August 25, 2018].

[Pur16] Pureswaran, Nair, Brody, IBM. Empowering the edge Practical insights on a decentralized Internet of Things. https://www-935.ibm.com/services/multimedia/GBE03662USEN.pdf/, 2016. [Online; accessed August 26, 2018].

[Quy15] Quynh H. Dang. Secure Hash Standard, National Institute of Standards and Technology, (NIST FIPS) - 180-4. https://dx.doi.org/10.6028/NIST.FIPS.180-4, 2015. [Online; accessed August 21, 2018].

[Rai17] Raiden Network Team. Raiden network documentation. https://raiden-network.readthedocs.io/en/stable/index.html, 2017. [Online; accessed 31-August-2018].

[RIC14] RICHARD GENDAL BROWN. A SIMPLE EXPLANATION OF BITCOIN "SIDECHAINS". https://gendal.me/2014/10/26/a-simple-explanation-of-bitcoin-sidechains/, 2014. [Online; accessed August 25, 2018].

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, pages 371–388, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[Sha17] Shaan Ray. What is Proof of Stake? https://hackernoon.com/what-is-proof-of-stake-8e0433018256, 2017. [Online; accessed August 25, 2018].

[She14] Shen Noether, Sarang Noether, Monero Research Lab. Monero is Not That Mysterious. https://lab.getmonero.org/pubs/MRL-0003.pdf, 2014. [Online; accessed August 25, 2018].

[Sku17] Skuchain. SkuChain Products. http://www.skuchain.com/#products, 2017. [Online; accessed August 25, 2018].

[Vit15a] Vitalek Buterin. A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2015. [Online; accessed 25-August-2018].

[Vit15b] Vitalik Buterin. Merkling in Ethereum. https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/, 2015. [Online; accessed August 31, 2018].

[Wik18a] Wikipedia. Iota (kryptowährung) — wikipedia, die freie enzyklopädie, 2018. [Online; Stand 22. August 2018].

[Wik18b] Wikipedia contributors. Blockchain — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=855030838, 2018. [Online; accessed 17-August-2018].

[Wik18c] Wikipedia contributors. Directed acyclic graph — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Directed_acyclic_graph&oldid=846566257, 2018. [Online; accessed 22-August-2018].

[Wik18d] Wikipedia contributors. Microtransaction — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Microtransaction&oldid=852232329, 2018. [Online; accessed 25-August-2018].

[Wik18e] Wikipedia contributors. Public-key cryptography — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Public-key_cryptography&oldid=853882522, 2018. [Online; accessed 22-August-2018].

[Wil12] Wilma Woo. CRYPTOCURRENCY IS HALF A TRILLION. https://bitcoinist.com/cryptocurrency-half-trillion-joint-market-cap-hits-500-billion/, 2012. [Online; accessed August 26, 2018].

[XWS+17] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba. A taxonomy of blockchain-based systems for architecture design. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 243–252, April 2017.

A Appendix Stuff

the Appendix

B Acronyms

DDOS Distributed Denial of Service

Dapps Decentralized Applications

Altcoins All other coins besides Bitcoin

JIT Just In Time

IoT Internet of Things

SCM Supply Chain Management

DLT Distributed Ledger Technology

TX Transaction

TX Transaction

POW Proof of Work

POS Proof of Stake

TPS Transactions per second

MTX Microtransactions

BTC Bitcoin

PoR Proof of Replication

PoS Proof of Storage

TTL Time To Live

IPFS InterPlanetary File System

HTTP Hyper Text Transport Protocol

FTP File Transfer Protocol