Università degli Studi di Camerino

School of Science and Technology

Master Degree in Computer Science (LM-18)

Privacy and blockchain: towards enabling confidential smart contracts execution

Candidate
Jerusha Josine Manchala
Student ID: 110582

Advisor
Leonardo Mostarda

Co-Advisor
Emanuele Scala

A.Y. 2020/2021

Abstract

Recently, enterprises have been seeing an increasing interest in blockchain and are beginning to investigate possible industry use cases. In this context, the question of blockchain begin to scale, they often run into data availability problems and some privacy issues. The different distributed applications beyond cryptocurrencies can be deployed on top of blockchain. One of these applications is smart contracts.

Ethereum is currently the most common blockchain platform for developing smart contracts. However, in recent times Hyperledger Besu was introduced as an open-source Ethereum client that runs on Ethereum public network, private networks and tests the network. Besu uses a private transaction manager TESSERA to implement privacy. This thesis research is meant to give a brief discussion about Ethereum, smart contracts, blockchain privacy and study of the Tessera system for privacy-preserving smart contracts.

We propose a detailed description of Public ledgers and Blockchain data privacy and possible privacy solutions and a Centralized blockchain, Ethereum, smart contracts and its privacy issues and solutions, Hyperledger Besu and Tessera . Information is retrieved from an official documentation.

Keywords: Blockchain, Ethereum, Privacy, Security vulnerabilities, Attacks, Smart contracts, privacy solutions, Hyperledger Besu, Tessera.

An investment in knowledge pays the best interest.

- Benjamin Franklin

Ringraziamenti

I would take this opportunity to express my sincere thanks and gratitude to my supervisor, Prof. Leonardo Mostara as well as co-supervisor Emanuele Scala, who gave me the golden opportunity to do this thesis on the topic (Privacy and blockchain : towards enabling confidential smart contracts execution), which helped me in doing a lot of research and I came to know about so many new things, I am really thankful for their vital support and guidance in completing this thesis. It is my honor to be one of their students.

I would also like to extend my gratitude and thanks to all the distinguished professors in the department of Computer Science. It was a wonderful opportunity for me to learn the advanced professional knowledge. I believe I was the luckiest one to meet a group of an amazing computer science friends who has helped me a lot not only in study, but also in the daily life generously. Last but not least. I would like to express my gratefulness to UNICAM and the City of Camerino. It's been an extraordinary two-year journey and experience in my life. It is my privilege to study in such an international, dynamic campus and I am grateful that I have met so many amazing and wonderful people and been friends with them. I will never forget the time spent in Camerino, Italy, this ancient town has brought some of the most beautiful memories to me from every single corner in my journey.

Thank You.

Index

List of Figures

1. Introduction

The decentralized application plays a remarkable and significant role in all aspects of our lives. A potential rise in interest in the blockchain in the upcoming years is expected in the fields of healthcare, supply-chain, logistics, industries, smart cities and smart homes etc.

In this chapter, I introduce knowledge of the blockchain in Section 2.1 classification of blockchain and all the technical aspects of Smart Contracts; in Section 3.1, I investigate and analysis blockchain and smart contracts privacy and security challenges and solutions; our research motivations and facing challenges is distributed in Section 4.4, Section 4.6, I briefly concluded our contribution to private transaction manager, and cryptocurreny research domain; finally the overall structure of this research is described in the Section 5, I have referenced and summarized the work of our predecessors in this domain.

2. Background

A blockchain is simply a cryptographically verifiable list of data using various techniques such as mathematics, algorithms, cryptography, economic models, etc. Blockchain is a public ledger of all cryptocurrency transactions that are digitalized and decentralized.

The bitcoin system uses the blockchain as its distributed public ledger, which records and verifies all bitcoin transactions on the open Bitcoin peer-to-peer networked system. A remarkable innovation of the bitcoin blockchain is its capability to prevent double spending time, with no reliance to any trusted central authority.

Despite the fact that, blockchain likewise has some security issues like public record and blockchain information protection and incorporated blockchain. The distinctive dispersed applications past digital forms of money can be sent on top of blockchain.

One of these applications is shrewd agreements, which are executable codes that work with, execute and uphold an arrangement between untrusted parties.

Ethereum is currently the most common blockchain platform for developing smart contracts. However, in recent times Hyperledger Besu was introduced as an open-source Ethereum client that runs on Ethereum public network, private networks and tests the network. Besu uses a private transaction manager TESSERA to implement privacy.

2.1 The introduction of blockchain

Blockchain is a common, permanent record that works with network. A resource can be tangible(a house, vehicle, money, land) or intangible(intellectual property, licenses, copyrights, marking). For all intents and purposes anything of significant worth can be followed and exchanged on a blockchain network, lessening hazard and reducing expenses for all included.

2.2 Why blockchain is important?

Business runs on data. The quicker it's gotten and the more precise it is, the better. Blockchain is great for conveying that data since it gives prompt, shared and

totally straightforward data put away on an unchanging record that can be gotten to exclusively by permissioned network individuals. A blockchain organization can follow orders, installments, records, creation and significantly more. Also on the grounds that individuals share a solitary perspective on reality, you can see all subtleties of an exchange start to finish, giving you more prominent certainty, just as new efficiencies and openings. [32].

Benefits of blockchain Tasks frequently squander exertion on copy record keeping and outsider approvals. Record-keeping frameworks can defenseless against extortion and cyberattacks. Restricted straightforwardness can slow information check. What's more with the appearance of IoT, exchange volumes have detonated. All of this eases back business, depletes the main concern - and implies we really want a superior way.

How Blockchain Works?

The main working processes of blockchain are as follows:

1. The sending node records new data and broad casting to network.
2. The receiving node checked the message from those data which it received, if the message was correct then it will be stored to block.
3. All receiving node in the network execute proof of work (PoW) or proof of stake (PoS) algorithm to the block.
4. The block will be stored into the chain after executing consensus algorithm, every node in the network admit this block and will continuously extend the chain base on this block.

The blockchain is an assortment of blocks, which is thoroughly open and public to everybody. The open record in the blockchain is dispersed in nature. The significant component of blockchain is that once the information is recorded into the record, then, at that point, that information can't be deleted. Each block present in that chain comprises of the information, hash to that specific information and the past hash. The information recorded in the blockchain relies upon the sort of the blockchain. In the event that the blockchain is identified with bitcoins, it will store information for exchanges, the data about the sender and recipient and the quantity of bitcoins present in the organization. Each block in the chain is having a hash esteem that can measure up to the fingerprints. As the new block is made, the hash of that specific square will likewise be produced. The hash of the square will be made with the adjustments made in the square. Subsequently, the hash esteems are a vital figure while making adjustments the block. Assuming that the hash upsides of any square will be changed, then, at that point, it won't be viewed as in a similar block. Other than the hash of the current block, the block additionally holds the hash of the past block. This assists with making a chain by connecting the current block to the past block. These highlights of a block in the chain makes blockchain safer [34].

Utilizing the procedure of Proof of Work, the formation of the new block gets delayed down dependent upon some degree. For this situation of bitcoin, the estimation

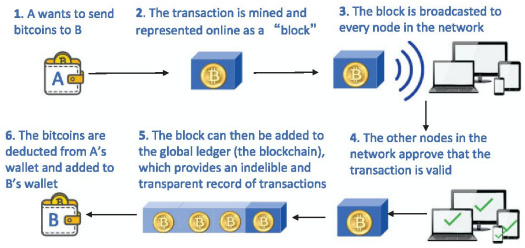

Fig. 1. How the Blockchain works.

Figure 2.1: How the blockchain works

of evidence of work requires almost 10 min to add the new block in the chain. This method upgraded the security in the blockchain. Since, in such a case that somebody will attempt to alter any block in the chain then he needs to recalculate the confirmation of work for every one of the accompanying block which are very troublesome. One of the significant benefits of blockchain nature, which makes blockchain secure themselves. Rather than brought together process for dealing with the chain, blockchain utilizes shared organization as the blockchain is open and public, anybody can join the organization. Subsequent to joining, all the member will get the total duplicate of the chain. By utilizing hash and confirmation of work is a serious secure instrument for blockchain.

The blockchain is evolving day by day. The smart contracts are the most recent development of the blockchain. The smart contacts are used to transfer coins among the nodes in the network automatically on the basis of some conditions are recorded in the blockchain.

2.3 Classification of Blockchain

Blockchain can be ordered into three significant classifications, each with various qualities and advantages. The engineering of a blockchain is vigorously reliant upon the entrance privileges gave to the network participants. [3]

2.3.1 Public Blockchain

Public blockchains are open-source information base accessible to everything network members, can peruse the information put away in the public blockchain and take an interest in the agreement interaction for approving another information block. It is completely decentralized and straightforward since no members can handle or alter the records information. Furthermore, the members in open blockchains are unknown, which ensures the member's security. Every member holds a duplicate of the blockchain and shoulders the obligation of approving the information put away in blockchain. A consensus is reached when all network participants have validated the data and processes are achieved by employing consensus algorithms.

2.3.1.1 Proof-of-work

PoW is the primary public blockchain agreement component created and was initially utilized by Bitcoin [4]. The PoW cycle is otherwise called mining. The members are known as miners, to contend with one another to check legitimacy of new information block and further add new block to the blockchain by addressing complex riddles. Just the principal member settling the riddle turns into the validator and will additionally add the information to the blockchain, , the participant who is chosen as the validator will be rewarded and the scalability of PoW is one of most popular consensus mechanisms. However, PoW becomes quite vulnerable if one participant holds at least 51% of the computational power around the blockchain network [4]. This is because the one

participant who holds at least 51% of the computational power can solve the complex puzzle more quickly than others and thus monopolize the rights for validating new data blocks. To solve the problems, it takes the significant time for participants to solve the complex puzzles and achieve a consensus.

2.3.1.2 Proof-of-stake

PoS was proposed to overcomes the drawbacks of PoW, Particularly the problem with high energy consumption. Similarly, only one participant is chosen as the validator to add the new block to the blockchain. In PoS random process is used to determine the participant who wins the privilege of becoming the validator. The participants who stake their cryptocurrencies are the candidate validators and a random process is utilized to select the validator from these candidate validators. Finally, the chosen validator will add the data lock to the highest chance of becoming the validator. Similar to PoW , the validator will also be rewarded with cryptocurrency. Since participants are not required to solve the complex puzzles in PoS. However, in PoS, the rich stakeholders are more likely to be selected as validators. Therefore, it is unfair for the less wealthy network participants.

2.3.2 Private Blockchain

In contrast to public blockchains, private blockchains are usually held and governed by a managing entity and only provide access to certified and trusted participants. Private blockchains allow participants to manage the data without revealing them to the public. Proof-of-Authority (PoA) is the most frequently used consensus mechanism in private blockchain. In PoA, the entity managing the private blockchain will select several trusted participants as validators. The new block is verified and then added to the blockchain by the chosen validators. PoA mechanism are less decentralized than mechanism such as PoW. If a chosen validator fails to accurately verify the data block, the blockchain managing entity can penalize validators for failing to carry out their duties.

2.3.3 Consortium Blockchain

Consortium blockchains are known as integration of private and public blockchains. Similar to private blockchains, consortium blockchains only allow authorized entities to write data participate in the consensus program. The data stored in consortium blockchains can be categorized as private data and public data. Private data can only be accessed by the managing entities, while public data can be assessed by all authorized network participants. Participants can control what data should be kept private and public. Consortium blockchains are generally used in the business industry to record cross-or-generational business transactions.

The original Blockchain Here some of the information about blockchain that was originally proposed.

- **Why Blockchain was invented:** The invention of the blockchain for bitcoin it the first digital currency to solve double-spending problem without the need of a trusted authority or central server... The blockchain is considered a type of payment rail [36].

- **What is the purpose of blockchain?** The goal of the blockchain is to allow digital information to be recorded and distributed, but not edited.

- **what are smart contracts?** Smart contracts are simple programs stored on a blockchain that run when predetermined conditions are met. That typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.

- **How smart contracts are used?** Smart contracts are a type of Ethereum account. This means they have a balance and they can send transactions over the network. However they're not controlled by a user, instead they are developed to the network and run as programmed. User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can define rules, like a regular contract, and automatically enforce then via the code. Smart contracts can not be deleted by default, and interactions with them are irreversible.

- **What is Tessera and how can we use it?** Tessera is an open-source private transaction manager developed under the Apache 2.0 license and written in java. The primary application of Tessera is as the privacy manager for privacy-enabled Ethereum clients such as GoQuorum [41] and Hyperledger Besu [51]. We can use it to generates and maintains private/public key pairs, self manages and discovers all nodes in the network and provides an API for communicating between Tessera nodes and an API for communicating with privacy-enabled Ethereum clients.

- What features Tessera has: Tessera is the Privacy manager for the GoQuorum and Hyperledger Besu Ethereum clients. Tessera's transaction manager:

  - Creates a peer to peer network with other transaction managers.

  - Delegates key management and payload encryption/decryption to the enclave.

  - Stores and retrieves saved data from the database.

  - Distributes private transaction payloads for privacy-enabled Ethereum clients.

2.3.4 What can we do with Tessera?

Tessera is a stateless Java System that is used to enable the encryption, decryption, and distribution of private transactions for Quorum and/or Besu.

Each Tessera node:

- Generates and maintains a number of private/public key pairs.
- Self manages and discovers all nodes in the network (i.e their public keys) by connecting to as few as one other node.
- Provides private and public API interfaces for communication.
- Private API - This is used for communication with Quorum.
- Public API - This is used for communication between Tessera peer nodes.

2.4 Smart Contracts

Generally speaking, smart contracts eliminate the need for third party enforcement by encapsulating contractual obligations in software, to be autonomously verified and enforced. Anyone with access to the contract can verify that a specific event will always result in a specific outcome. While it is expected that smart contracts maybe used in the future to replace many kinds of real-world paper contracts, to date the most common use case for smart contracts is to create virtual tokens inherit many of the properties of the distributed system in which they are created. [20]

A network powered by Tessera that enables private payloads and dual transaction states has the potential for creating a special off-chain testing environment specifically for proprietary software acting as payloads. Only the involved parties will have access to the payload being tested, and the entire verification process can be automated by a smart contract. They can allow or restrict participants from executing certain functions, and can restrict access to the network itself.

Putting It All Together

Smart contracts can be utilized to fill roles in an extraordinary assortment of ventures. Regardless of whether administrative consistence, authoritative implement capacity, cross-line monetary exchanges, property proprietorship, home purchasing, supply the executives, material provenance, archive the board and numerous different applications.

Supply Chain Management: Making supply chains more straightforward by means of smart contracts is assisting with streamlining the development of

merchandise and reestablish trust in exchange. smart contracts can record possession freedoms as things travel through the inventory network, affirming who is liable for the item at some random time. The completed item can be checked at each phase of the conveyance interaction until it arrives at the client.

Insurance:

Smart contracts could also be used in the insurance sector. This sector nowadays lack automated administration. It can take months for an insurance claim to be processed and paid. Smart contracts can simplify and streamline the process by automatically triggering a claim when certain events occur. Specific details could thereby be recorded on the blockchain in order to determine the exact amount of compensation.

Financial Industry:

The most far reaching utilization of smart contracts stays in the monetary business, as cash and going with records become electronic. In the monetary administrations area the chances for brilliant agreements incorporate, for instance, installment handling, clearing/repayment of monetary instruments, exchange finance, just as administrative innovation, for example, smoothed out 'know your customer' certification.

Real Estate:

It has lot of risks as well as time taking.It can likewise go through various phases of legitimate activity are additionally need a ton of paper signings just as shared check of the archives. Blockchain innovation and a brilliant agreement can conquer the issue related with land area. A concentrated framework can permit purchasing just as sell properties without the third party. The documents are also stored digital ledger distributed database whenever everyone can see.

Digital Right Management:

Digital right industry involves multiple parties to create an event. The percentage of payment and copyright is one of the issues arises. Using smart contract-based system guarantee that royalties go to actual recipient by using ownership right in blockchain system.

Internet of Things:

This is one of the most promising areas of research. The IoT devices are resource constraint device having less memory power as well as less processing power. The blockchain technology based smart home, smart city, smart transportation, smart monitoring of environment application is already done some research. If smart contract concept integrates with the blockchain system the IoT becomes more autonomous.

Smart Contracts in Ethereum: A smart contract is an "autonomous agent" sorted in the blockchain, encoded as part of a "creation" transaction that introduces a contract to the blockchain. When effectively made, a smart contract is characterized by a "contract address"; each agreement holds some measure of virtual coins (Ether), has its own private storage, and is associated with its predefined executable code. A contract states consists of two main parts: a private storage and the

amount of virtual coins (Ether) it holds (called balance). Contract code can control factors like in conventional basic projects. The code of an Ethereum contract is a low-level, stack based bytecode language alluded to as Ethereum Virtual Machine (EVM) code. Clients characterize contracts utilizing undeniable level programming dialects, e.g., Solidity (a JavaScript-like language), which are then arranged inti EVM code. To invoke a contract at address, users send a transaction to the contract address. A transaction typically includes: payment to the contract for the execution in Ether and or input data for the invocation.

2.5 Technical Aspects Of Smart Contracts

This scientific research on blockchain-based contracts mesmerized on different directions of the technical specialization. Stared as a minimal decentralized immutable ledger, the blockchain technology has emerged with vital features such as improved security, scalability and optimal operation thanks to the contribution of the research conducted world- wide. The significant technical aspects of the blockchain-based smart contracts are discussed in this section ans shows a quick overview of technical aspects of smart contracts.

A.SECURITY ATTACKS, VULNERABILITIES AND POSSIBLE SOLUTIONS Smart contracts are beginning to be widely adopted in many industry use cases. Security is of vital importance in consideration of smart contracts as if affects the functionality of entire blockchain. The various attacks on the smart contracts, existing research over these attacks, and potential solutions are discussed in this section. Due to the wide adoption of Ethereum smart contracts in real- life use cases and security attacks occurred in recent years, in this section we focus only Ethereum to discuss the attacks and vulnerabilities of smart contracts [43]

1) Different Attacks and Vulnerabilities: There exist multitudinous attacks in the smart contract context [context from]. These assaults lead due to various like programming errors, limitations in the programming dialects, and security provisos. The consequences of these assaults comprise of numerous calculations for the blockchain network and their exactness, misfortune if local cryptographic money and end the accessibility of the framework. The assaults can be recognised on both public and private blockchain stages. The role of smart contracts and their precision can be more momentous in the public blockchain than in private blockchain. The debugging or any correction is cumber-stone interaction since the contracts adopted to all nodes in a blockchain network.

Re-entrency vulnerability (A1): In general terms, the re-entrancy vulnerability exploited when one smart contract invokes another smart contract iteratively and the smart contract that initiates the invocation is malicious. Such attacks were ample in the Ethereum platform. In the long run, the summoning smart contract's Ethers are moved to the malicious contract's account. The DAO assault, a mysterious hacker stole 50M USD worth of Ethers from the 168M invested through a virtual venture capital raising in 2016. A coder discovered some escape clause, when a split capacity is summoned, the code retrieves Ether first and update balance later and not checking whether is it a recursive call. The attacker recursively calls split capacities

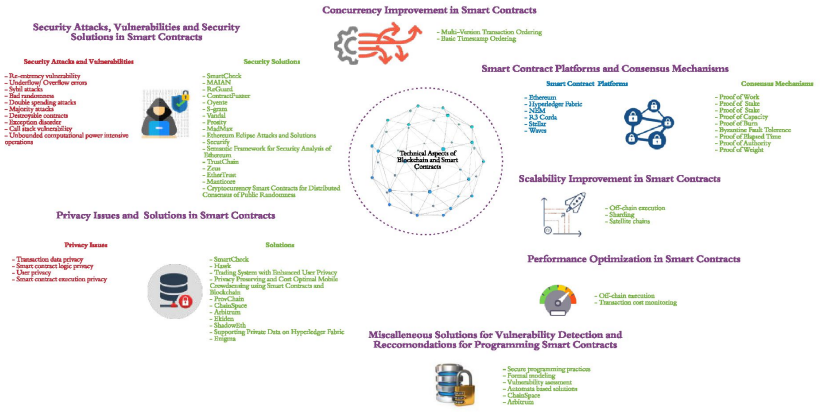

Figure 2.2: Overview of the technical aspects of smart contracts

and recovers their funds before the code actually looks at the equilibrium.

Underflow/Overflow errors (A2): Underflow or Overflow happened when the aftereffect of a specific math activity was either less or more than the base or most extreme numeric information type used in the smart contract platform. The Ethereum stage is utilizing uint256 [26]. Conceptually, the Ether equilibrium of 0 can be changed into the most extreme worth of the unit256 or Ether equilibrium of greatest worth can change into 0. Nonetheless, the software engineers are needed to think about such conditions when coding the contract [33].

Sybil attacks (A3): Sybil attack [49] occurs when a member attempts to take over the peer network by conceiving fake identities explicitly. Such circumstances [13] will prompt an unbalanced control of the network which can prompt the capturing of the disseminated blockchain peer environment. For example, assuming that the agreement depends on the greater part casting a ballot of a specific blockchain, the individuals expressly made on the assault can assume control over the agreement of the network. The framework is inclined to such a danger when the onboarding system to the blockchain is exceptionally insignificant and computerized.

Bad randomness (A4): Randomness is required in the smart contracts, particularly in gaming and betting setting. Also, there are utility capacities which require randomness [7]. The approaches for random or pseudorandom number generation will be vulnerable in some circumstances [15]. The usage of block variables and block hashes as sources of entropy can be vulnerable.

Double spending attacks (A5): The spending of the native token is common in almost all smart contract platforms [23].In every exchange, there is a risk that a similar client can spend a specific symbolic at least twice. This kind of assaults is known as the double spending attacks [65].

Majority attacks (A6): Majority of attacks occur when some malicious users or groups take over the control over the control to revise exchange history or keep new exchanges from affirming [66]. The assault can happen when a specific blockchain consortium embraced with greater part casting a ballot agreement [46]. Contingent upon the user necessity, at times block mining and exchange check is offloaded to some committed driving companions of every individual from the consortium. If the majority of leading peers are hijacked by the malicious user, the similar circumstance occurred.

Destroyable contracts(A7): The self-destruct vulnerabilit removes the content of a smart contract by deleting the bytecode at the particular addresses. Furthermore, it sends all contract's funds to a specific target address, making the contract non-functional.

Exception disorder (A8): Exception disorder occurs due to the exceptions lead to a failure when they were not properly handled . Special case dealing with is a significant practice in programming, either blockchain or some other context. The improper handling of exceptions, especially when one contract invokes another, affects the operability of the entire network [27]. Hence, the exception disorder is highlighted as a major vulnerability.

Call stack vulnerability (A9): The call stack's depth is limited up to a

certain value in the execution environment of the smart contract. For instance, in Ethereum the call stack's depth limited to 1024 frames. The operation fails when the depth of the call reaches the limitation. This can occur due to various reasons such as programming errors [38].

Unbounded computational power intensive operations (A10): Each operation on the smart contracts requires consumption of computational power [54]. For instance, the cost for the computational power in Ethereum is called Gas. The gas utilized to evaluate the computational resource consumption of a particular operation on the smart contract. Unbounded and unrestricted computational power intensive operations lead to various errors and eventually affect the system [57].

2.6 Programmable Smart Contracts

- Ethereum is a decentralized virtual machine, which runs programs called contracts. Contracts are written in a Turing-complete bytecode language, called Ethereum Virtual Machine (EVM) bytecode. A contract is conjured by considering one of its capacities, where each function is characterised by a grouping of guidelines. The contract keeps a tenacious inside state and can receive(transfer) money from (to) clients and other contracts.Users send exchanges to the Ethereum network to summon capacities. Every exchange might contain input boundaries for the contract and an associated monetary amount, possibly 0, which is transferred from the user to the contract.

- After getting an exchange, the contract gathers the cash shipped off it, executes a capacity as per in-out boundaries, and updates its inside state. All exchanges are recorded on a decentralised ledger, called blockchain. A grouping of exchanges that start from the making of the network exceptionally decides the condition of each agreement and counterbalances of users and contracts. The blockchain doesn't depend on a confided in focal power, rather, every exchange is handled by a huge network of commonly untrusted peer called miners. Users constantly broadcast transactions to the network. Miners add transactions to the blockchain via a proof-of-work consensus protocol.

-Transaction fees. In exchanges for including her transactions in the blockchain, a user pays transaction fees to the miners, proportionally to the execution time of her transaction. This reality could somewhat influence the money related investigation of the client gain, yet could likewise present bugs in a program, as there is a bound on execution time that can't be executed. Subsequently, it is conceivable that a few capacities would never be called, or surprisingly more terrible, a user could effectively give input that would prevent other from invoking a certain function.

-Recursive invocation of contracts. A contract function could invoke a function in another contract, which in turn can have a call to the original contract. The underling Ethereum semantic in recursive invocation was the root cause for the notorious DAO hack [42].

-Behaviour of the miners. Previous works have suggested that smart contracts could be implemented to encourage miners to deviate from their honest behavior. This could in theory introduce bugs into a contract, e.g., a contract might give unfair advantage for a user who is a big miner.

2.6.1 Tokens and User Utility

A user's utility is controlled by the Ether she spends and gets, however could likewise be impacted by the condition of the contract. Most remarkably, smart contracts are utilised to give tokens, which can be considered a stake in an organisation or an association, consequently to an Ether (or tokens) venture (see a model in Fig.1). These tokens are adaptable among clients and are exchanged trades consequently to Ether, Bitcoin and Fiat cash. At the hour of composing, smart contracts start up tokens worth billions of dollars . Subsequently, acquiring or losing tokens has clear utility for the user. At a bigger degree, user utility could likewise be impacted by more theoretical stockpiling changes. A few clients would pay to have an agreement proclaim them as Kings of Ether, while others could acquire from enlisting their space name in a smart contract storage.

```
1  contract Token {
2          mapping(address=>uint) balances;
3          function buy() payable  {
4                balances[msg.sender] += msg.value;
5          }
6          function transfer( address to, uint amount ) {
7             if(balances[msg.sender]>=amount) {
8                balances[msg.sender] -= amount;
9                balances[to] += amount;
10             }
11       }
12    }
```

Fig.1. Token contract example.

2.7 Programming Language for Smart Contracts

In this section we present our programming language for smart contracts that supports concurrent interactions between parties. A party denotes an agent that decides to interact with the contract. A contract is a tuple $C = (N, I, M, R, X_0, F, T)$ where $X := N \cup I \cup M$ is a set of variables, R describes the range of values that can be stored in each variable, X_0 is the initial values stored in variables, F is a list of functions and T describes for each function, the time segment in which it can be invoked. We now formalize these concepts.

Variables. These are three distinct and disjoint types of variables in X:

- N contains "numeric" variables that can store a single integer.

- I contains "identification" ("id") variables capable of pointing to a party in the contract by her address or storing NULL. The notion of ids is quite flexible in our approach: The only dependence on ids is that they should be distinct and an id should not act on behalf of another id. We simply use different integers to denote distinct ids and assume that a "faking of identity" does not happen. In Ethereum this is achieved by digital signatures.

-M is the set of "mapping" variables. Each m ∈ M maps parties to integers.

Bounds and Initial Values. The tuple $R = (\underline{R}, \overline{R})$ where $\underline{R}, \overline{R} : N \cup M \to \mathbb{Z}$ represent lower and upper bounds for integer values that can be stored in a variable. For example, if $n \in N$, then n can only store integers between $\underline{R}(n)$ and $\overline{R}(n)$ Similarly, if $m \in M$ is a mapping and $i \in I$ stores an address to a party in the contract, then m[i] can save integers between $\underline{R}(m)$ and $\overline{R}(m)$. The function $X_0 : X \to \mathbb{Z} \in \{NULL\}$ assigns an initial value to every variable. The assigned value is an integer in case of numeric and mapping variables, i.e., a mapping variable maps everything to its initial value by default. Id variables can either be initialized by NULL or an id used by one of the parties.

Functions and Timings. The sequence $F =< f_1, f_2, ..., f_n >$ is a list of functions and $T = (\underline{T}, \overline{T})$, where $\underline{T}, \overline{T} : F \to \mathbb{N}$. The function f_i can only be invoked in time-frame $T(f_i) = [\underline{T}(f_i)\ , \overline{T}(f_i)]$. The contract used a global clock, for example the current block number in the blockchain, to keep track of time.

2.7.1 Syntax

We provide a simple overview of our contract programming language. Our language is syntactically similar to Solidity [56] , which is a widely used language for writing Ethereum contracts. A translation mechanism for different aspects. An example contract, modeling a game of rock-paper-scissors, is given in Fig. 2. Here, a party, called issuer has issued the contract and taken the role of Alice. Any other party can join the contract by registering as Bob and then playing rock-paper-scissors. To demonstrate our language, we use a bidding mechanism.

Declaration of Variables. After the variables, the functions are defined one-by-one. Each function begins with the keyword function followed by its name and

```
1  contract RPS {
2  map Bids[0, 100] = 0;
3  \id Alice = issuer;
4  \id Bob = \null;
5  numeric played[0,1] = o;
6  numeric AliceWon[0,1] = 0;
7  numeric BobWon[0,1] = 0;
8  numeric bid[0, 100] = 0;
9  numeric AliceMove[0,3] = 0;
10 numeric BobMove[0,3] = 0;
11 // 0 denotes no choice,
12 //1 rock, 2 paper,
13 //3 scissors
14
15 function registerBob[1,10]
16     (payable bid : caller) {
```

```
17      if(Bob==\null)
18      {
19          Bob = caller;
20          Bids[Bob]=bid;
21     }
22      else{
23        payout(caller, bid);
24   }
25   }
26  function play[11, 15]
27  (AlicesMove:Alice = 0,
28  BobsMove : Bob = 0,
29  payable Bids[Alice]: Alice) {
30    if (played==1)
31       return;
32    else
33       played = 1;
34    if(BobsMove==0 and AlicesMove!=0)
35          AliceWon = 1;
36    else if(AlicesMove==0 and Bobsmove!=0)
37          BobWon = 1;
38    else if(AlicesMove==0 and BobsMove==0)
39    {
40          AliceWon = 0;
41          BobWon = 0;
42    }
43
44    else if(AlicesMove==BobsMove+1 or
45       AlicesMove==BobsMove-2)
46          AliceWon = 1;
47    else
48          BobWon = 1;
49  }
50   Function getReward[16,20] () {
51   if(caller==Alice and AliceWon==1
52   or caller==Bob and BobWon==1)
53      {
54
55      Payout(caller,
56              Bids[Alice]  +  Bids[Bob]);
57      Bids[Alice] = 0;
58      Bids[Bob] = 0;
59      }
60    }
61   }
```

Fig.2. A rock-paper-scissors contract.

the time interval in which it can be called by parties. Then comes a list of input parameters. Each parameter is of the form variable : party which means that the designated party can choose a value for that variable. The chosen value is required to be in the range specified for that variable. The keyword caller denotes the party that has invoked this function and payable signifies that the party should not only decide a value, but must also pay the amount she decides. For example, registerBob can be called in any time between 1 and 10 by any of the parties. At each such invocation the party that has called this function must pay some amount which will be saved in the

variable bid. After the decisions and payments are done, the contract proceeds with executing the function.

Types of Functions. There are essentially two types of functions, depending on their parameters. *One-party functions*, such as **registerBob** and **getReward** require parameters from caller only, while *multi-party functions*, such as play ask several, potentially different, parties for input. In this case all parties provide their input decisions and payments concurrently and without being aware of the choices made by other parties, also a default value is specified for every decision in case a relevant party does not take part.

Summary. Putting everything together, in the contract specified in Fig. 2, any party can claim the role of the Bob between 1 and 10 by paying a bid to the contract, if the role is not already occupied. Then at time 11 one of the parties calls **play** and both parties have until time 15 to decide which choice (rock, paper, scissors or none) they want to make. Then the winner can call **getReward** and collect her prize.

2.7.2 Semantics

In this section we present the details of the semantics. In our programming language there are several key aspects which are non-standard in programming languages, such as the notion of time progress, concurrency, and interactions of several parties. Hence we present a detailed description of the semantics. We start with the requirements.

Requirements. In order for a contract to be considered valid, other than following the syntax rules, a few more requirements must be met, which are as follows:

- We assume that no division by zero or similar undefined behavior happens.
- To have a well-defined message passing, we also assume that no multi-party function has an associated time interval intersecting that of another function.
- Finally, for each non-id variable v, it must hold that $underline R(v) \leq X_0(v) \leq \overline{R}(v)$ and similarly, for every function f_i, we must have $\underline{T}(f_i) < \overline{T}(f_i)$.

Overview of Time Progress. Initially, the time is 0. Let F_t be the set of functions executable at time t, i.e., $F_t = f_i \in F | t \in T(f_i)$, then F_t is either empty or contains one or more one-party functions or consists of a single multi-party function. We consider the following cases:

- F_t *empty.* If F_t is empty, then nothing can happen until the clock ticks.

Execution of one-party functions. If F_t contains one or more one-party functions, then each of the parties can call any subset of these functions at time t. If

there are several calls at the same time, the contract might run them in any order. While a function call is being executed, all parties are able to see the full state of the contract, and can issue new calls. When there are no more requests for function calls, the clock ticks and the time is increased to t + 1. When a call is being executed and is at the beginning part of the function, its caller can send messages or payments to the contract. Values of these messages and payments will then be saved in designated variables and the execution continues. If the caller fails to make a payment or specify a value for a decision variable or if her specified values/payments are not in the range of their corresponding variables, i.e., they are too small or too big, the call gets canceled and the contract reverts any changes to variables due to the call and continues as if this call had never happened.

Execution of multi-party functions. If F_t contains a single multi-party function f_i and $t < \overline{T}(f_i)$, then any party can send messages and payments to the contract to specify values for variables that are designated to be paid or decided by her. These choices are hidden and cannot be observed by other participants. She can also change her decisions as many times as she sees fit.

The clock ticks when there are no more valid requests for setting a value for a variable or making a payment. This continues until we reach time $\overline{T}(f_i)$.At this time parties can no longer change their choices and the choices become visible to everyone. The contract proceeds with execution of the function. If a party fails to make a payment/decision or if Null is asked to make a payment or a decision, default behavior will be enforced. Default value for payments is 0 and default behavior for other variables is defined as part of the syntax. For example, in function play of Fig. 2, if a party does not choose, a default value of 0 is enforced and given the rest of this function, this will lead to a definite loss.

Given the notion of time progress we proceed to formalize the notion of "runs" of the contract. This requires the notion of labels, control-flow graphs, valuations, and states, which we describe below.

Labels. Starting from 0, we give the contract, beginning and end points of every function, and every command a label. The labels are given in order of appearance. As an example, see the labels in parentheses in Fig. 2.

Entry and Exit Labels. We denote the first (beginning point) label in a function f_i by $\square_i$ and its last (end point) label by $\boxtimes_i$.

Control Flow Graphs CFG_s. We define the control of lower graph $CFGi$ of the function. f_i in the standard manner, i.e., $CFG_i = (V, E)$, where there is a vertex corresponding to every labeled entity inside f_i. Each edge $e \in E$ has a condition *cond(e)* which is a boolean expression that must be true when traversing that edge.

Valuations. A valuation is a function val, assigning a value to every variable. Values for numeric variables must be integers in their range, values for identity variables can be party ids or Null and a value assigned to a map variable m must be a function val(m) such that for each identity i, we have $\underline{R}(m) \leq val(m)(i) \leq \overline{R}(m)$. Given a valuation, we extend it to expressions containing mathematical operations in the straight-forward manner.

States. A state of the contract is a tuple s = (t,b,l,val,c), where t is a time stamp, $b \in \mathbb{N} \cup 0$ is the current balance of the contract, i.e., the total amount of payment to the contract minus the total amount of payouts, l is a label (that is being executed), val assigns values to variables and $c \leq P \cup \perp$, is the caller of the current function. $c = \perp$ corresponds to the case where the caller is undefined, e.g., when no function is being executed. We use S to denote the set of all states that can appear in a run of the contract as defined below.

Runs. A run ρ of the contract is a finite sequence $\rho_j = (t_j, b_j, l_j, val_j, c_j)_r^j = 0$ of states, starting from $(0, 0, 0, X_0, \perp)$, that follows all rules of the contract and ends in a state with time-stamp $t_r > max f_i \overline{T}(f_i)$.

These rules must be followed when switching to a new state in a run:

- The clock can only tick when there are no valid pending requests for running a one-party function or deciding or paying in multi-party functions.

- Transitions that happen when the contract is executing a function must follow its control flow graph and update the valuation correctly.

- No variable can contain an out-of-bounds value. If an overflow or underflow happens, the closest possible value will be saved. This rule also ensures that the contract will not create new money, given that paying more than the current balance of the contract results in an underflow.

- Each party can call any set of the functions at any time.

Given a valuation, we extend it to expressions containing mathematical operations in the straight-forward manner.

Remark 1. Note that in our semantics each function body completes its execution in a single tick of the clock. However, ticks might contain more than one function call and execution.

Run Prefixes. We use H to mean the set of all prefixes of runs and denote the last state in $\eta \in H$ by $end(\eta)$. A run prefix η' is an extension of η if it can be obtained by adding one state to end of η.

Probability Distributions. Given a finite set X, a probability distribution on X is a function $\delta : X \to [0, 1]$ such that $\sum_x \in_X \delta(x) = 1$. Given such a distribution, its support, *Supp* δ, is the set of all $x \in X$ such that $\delta(x) > 0$. We denote the set of all probability distribution on X by $\triangle(X)$.

Moves. We use M for the for the set of all moves. The moves that can be taken by parties in a contract can be summarized as follows:

- Calling a function f_i, we denote this by call(f_i).

- Making a payment whose amount, y is saved in x, we denote this by *pay(x,y)*.

- Deciding the value of x to be y, we denote this by *decide(x,y)*.

- Doing none of the above, we denote this by $\boxtimes$.

Permitted Moves. We define $p_i : S \to \mathcal{M}$, so that $P_i(s)$ is the set of permitted moves for the party with identity i if the contract is in state $s = (t, b, l, val, p_j)$.

It is formally defined as follows:

- If f_k is a function that can be called at state s, then $call(f_k) \in P_i(s)$.

- If $l = \square_q$ is the first label of a function f_q and X is a variable that can be decided by **i** at the beginning of the function f_q, then $decide(x, y) \in P_i(s)$ for all permissible values of y. Similarly if x can be paid by $i, pay(x, y) \in P_i(s)$.

Policies and Randomized Policies. A policy π_i for party is i is a function $\pi_i : H \to A$, such that for every $\eta \in H, \pi_i(\eta \in P_i(end.(\eta))$. Intuitively, a policy is a way deciding what move to use next, given the current run prefix. A policy profile $\pi = (\pi_i)$ is a sequence assigning one policy to each party i. The policy profile π defines a unique run ρ^π of the contract which is obtained when parites choose their moves according to π. A randomized policy ξ_i for party i is a function $\xi_i : H \to \triangle(\mathcal{M})$, such that $Supp(\xi_i(s)) \subseteq P_i(s)$. A randomized policy assigns a probability distribution over all possible moves for party i given the current run prefix of the contract, then the party can follow it by choosing a move randomly according to the distribution. We use Ξ to denote the set of all randomized policy profile, Ξ_i for randomized policies of i and $\Xi_{-}i$ to denote the set of randomized policy profiles for all parities expert i. A randomized policy profiles ξ is a sequence (ξ_i) assigning one randomized policy to each party. Each such randomized policy profile includes a unique probability measure on the set of runs, which is denoted as $Prob^\xi[.]$. We denote the expectation measure associated to $Prob^\xi[.]$ by $\mathbb{E}^\xi[.]$

2.7.3 Conclusion:

In this background I present the main topics of this thesis, which plays a major role in the blockchain technology. There are several interesting directions of future work. First, I present the classification of blockchain. Secondly, smart contracts programming language with an example, thirdly privacy and security issues, finally the private transaction manager Tessera.

3. Methodology and Framework

To understand how decentralized applications achieve their goals, we need to lay the theoretical framework and explain the key concepts enabling this software development.

3.1 Cryptography

Cryptography is the review and practice of creating conventions to deliver a snippet of data mysterious. While this definition covers the fundamental piece of private correspondence essential for different conditions from the Antiquity to present day days, it is conventional. For the peruser to get a handle on the utility of cryptography for present day applications, the thought of a foe should be fused, as it was presented by Ron Rivest in 1990 [47]. Indeed, the modern application of cryptography includes the practice of keeping secret information away from adversaries.

There is a need here likewise to explain a portion of the standard terms utilized in the cryptography domain. The underlying condition of a message going to be communicated is known as the plaintext, while the encoded structure is known as the ciphertext. The most common way of changing over the plaintext to ciphertext is characterised as encryption. Then again, the task to change the ciphertext back to plaintext is portrayed as decryption. The limited grouping of steps that can be followed like a method, thusly the calculation needed to encode and unscramble a message, is generally alluded to as a code. Perhaps the least complex code known from history, alluded to as Caesar figure, was utilised by Julius Caesar for his private composed correspondence. Based on substitution, the cipher involved a simple shift by three positions upwards on the alphabet on the plaintext letters, as depicted in Figure 3.1. The reverse operation was performed for the decryption.

A key idea separating encryption from encoding is the presence of a key. A key is an extremely agreeing term utilised for a different snippet of data needed for encryption and decryption to be performed. Assuming an equal could be drawn between a cipher and a padlock, the cryptographic key is the actual key that allows its owner to lock and unlock the padlock without a hassle. As the accompanying segment will portray, the key is regularly produced utilising a particular cycle. The cipher refers to encryption and decryption algorithms, and the term cryptosystem describes both procedures and the key generation.

The web is an uncertain interchanges framework. The motivation behind why the web today is definitely safer than it was during the 1970s and the 1980s lies with the

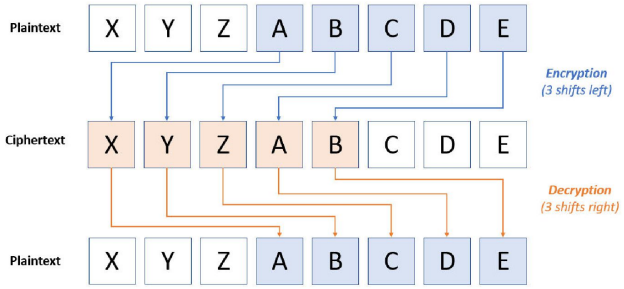

Figure 3.1: The caesar cipher

lengthy utilisation of cryptography. The conventional definition depicted in the past passage emerges in light of the fact that the threats are diverse, and cryptography gives ways of countering them effectively. At the point when we communicate information on the web, When we transmit data online, we need specific properties, and their role is to protect against a particular type of threat. These are:

- **Confidentiality:**

  The ability to protect from non-authorized disclosure of information. Confidentiality should be performed so that the actual information remains hidden from unauthorized parties, and its very existence shall not be revealed.

- **Integrity:**

  The protection from unauthorised alteration. The honesty of a message will be safeguarded and effectively certain by the beneficiary to affirm whether the genuine message payload has been altered during the transmission. This additionally incorporates unintentional information debasement because of commotion in the transmission station, regularly in broadcast communications.

- **Authentication:**

  The confirmation that the user or the element at the opposite finish of the correspondence channel is who they say they are. Realising the sender's personality additionally applies to information validation, the ability to verify that the party indeed sent the message we believe it was sent from and not an impostor or a spy.

- **Non-repudiation:**

  The ability to ensure that no parties can claim they did not interact in communication when they did. This includes both non- repudiation of destination: the denial of receiving a message and non- repudiation of origin: the denial of sending a message.

It is essential to clarify that many of the above properties are not achieved separately and more than often overlap with other. Similarly, encrypting a message to preserve confidentiality also provides integrity, as the decryption algorithm would require the message to be unaltered for the decryption to be successful.

3.1.0.1 Types of Cryptography

Moving forward to approach the blockchain, we need to describe more the various types of cryptography today. Those are Secret Key Cryptography (SKC), Public Key Cryptography (PKC), and Hash Functions.

3.1.0.2 Secret Key Cryptography(SKC)

Secret Key Cryptography uses a single key to encrypt and decrypt data. Both encryption and decryption utilize the same shared key used by both the sender and the receiver. The scheme depicting a secret key cryptosystem Figure 6 appears symmetrical, so this kind of cryptography is also referred to as symmetric key cryptography.

The challenge in this type of cryptography is the secure distribution of the secret key. The reason is that the only way to fortify the security of the information in a shared secret key cryptographic system is to protect the secrecy of the secret key. This need requires both participants to communicate through a secure channel before the cryptographic transmission takes place.

3.1.0.3 Public Key Cryptography (PKC)

Public Key Cryptography uses two numerically related keys. Knowing one of the keys doesn't empower somebody to decide the other key without any problem. One key is utilised to encrypt the plaintext and the other key to decrypt the ciphertext. There is no limitation on which key is applied first, however the significance is that both keys are needed all together for the cryptosystem to work. One of the keys is public, and its proprietor can unreservedly appropriate it to different gatherings. The other key is private and will be kept mystery and never uncovered.The employment of a pair of keys made this type of cryptography also be known as asymmetric cryptography. This asymmetry appears schematically below in Figure 3.3

Public Key Cryptography is considered to be the most significant development in cryptography in the past centuries. It was first described by Martin Hellman and Whitfield Diffie in 1976 [21]. It is primarily used today for authentication, nonrepudiation, and key exchange.

A notable application of Public Key Cryptography is the RSA cryptosystem, named out of Ronald Rivest, Adi Shamir, and Leonard Adleman, who invented it. Thousands of software products use RSA for key exchange, digital signatures, or encryption of small data blocks. Other well-regarded asymmetric key techniques for varied purposes include the Diffie Hellman key exchange protocol, the Digital Signature Algorithm (DSA), the ElGamal encryption system, and lots of developments in the field of Elliptic-curve cryptography (ECC).

3.1.0.4 Hash Functions

Hash Functions are algorithms that use no key. They are also known as message digests and one-way encryption. In this approach, one-way mathematical functions generate a fixed-length hash value from the input. The critical aspect is that while the hash functions are relatively easy to calculate, the output hash is very difficult to reproduce the initial input.

Hash functions can map an indeterminable size of plaintext into a fixed size

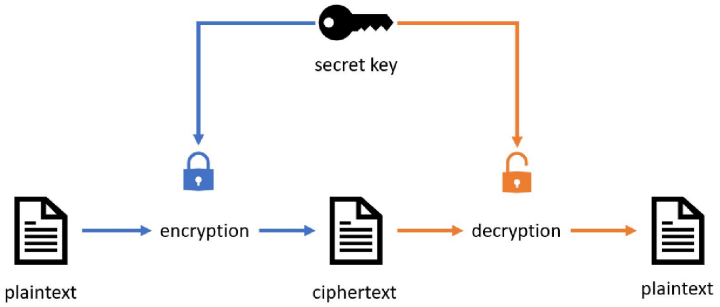

Figure 3.2: Secret Key Cryptography

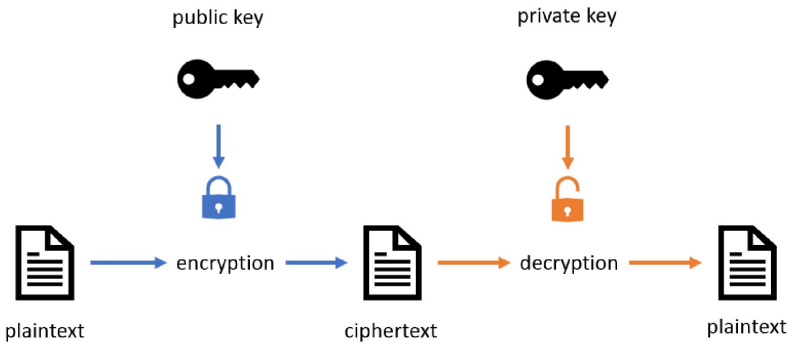

Figure 3.3: Public Key Cryptography

hash, and they are typically used to provide a digital fingerprint of data. A minor change in input will result in a completely different output, as shown in Figure 3.4 . Therefore, many systems employ hash algorithms to ensure the integrity of files or passwords. Hashing is also commonly used with digital signatures, as we will discover in the next section.

Famous hash algorithms include the Message Digest (MD) algorithms. However, the latest version, MD5, was identified as cryptographically broken and unsuitable for further use in 2008 . Popular alternatives include the Secure Hash Algorithms (SHA). Although specific variants of SHA-2 are still considered robust and secure, SHA-3 was standardized by The National Institute of Standards and Technology (NIST) in 2015 . SHA-3 is based on Keccak sponge functions.

3.1.1 Digital Signatures

Digital signatures are the public-key fundamentals of message authentication. It is common in the physical world to use handwritten signatures on written messages as they bind someone with a message. Equivalently, a digital signature is a method that binds a person to digital data. This binding can be verified by the recipient of the message or any other party.

In cryptography, a digital signature is a value calculated from the message and a secret key known only by the sender. Consequently, a digital signature scheme is heavily based on Public Key Cryptography (PKC). Such a scheme can be depicted in Figure 3.5

Here are the process steps in detail:

1. The signer passes data to a hash function and generates a hash.

2. The hash value is passed as input to the signature algorithm along with the signer's private key. The algorithm produces the digital signature for the given hash.

3. The signature is attached to the data and sent o the Verifier.

4. The Verifier uses the digital signature as input to the verification algorithm along with Signer's public key.The verification algorithm produces an output.

5. The verification relies on whether the hash value and the output of the verification algorithm are equal.

An observant reader would notice that a hash of the data was created instead of signing the original data using the signature algorithm. This decision lies in the efficiency of the scheme. As the data verified can grow quite large on many occasions, signing using a signing algorithm could be proved computationally expensive and slow. On the contrary, hash values are relatively small in length while guaranteeing data integrity.So, encrypting a hash is far more efficient than encrypting the original data.

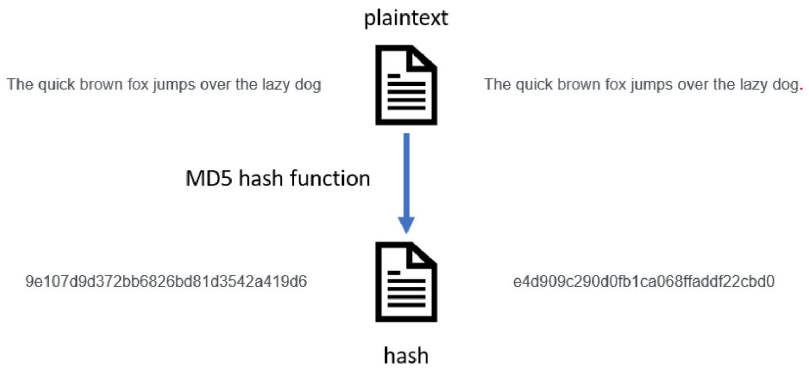

Figure 3.4: Showcase of MD5 hash function

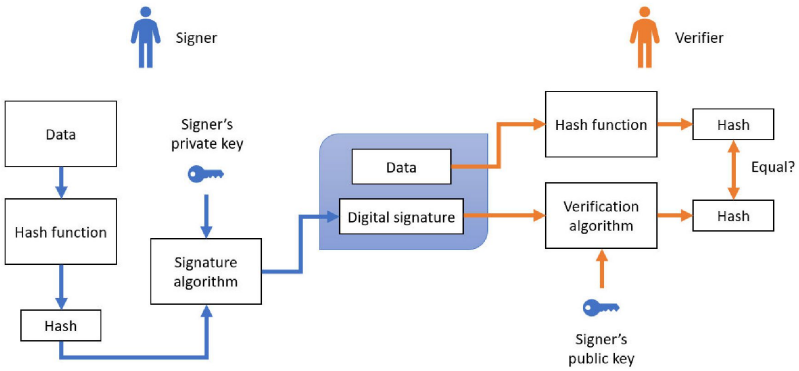

Figure 3.5: A digital signature scheme

3.2 Blockchain

After we have covered the underlying fundamentals of cryptography and cryptographic concepts, we can proceed to a significant notion of this dissertation. Blockchain, a disruptive technology, is considered the basis for providing any reliable and secure decentralized solution.

The notion of blockchain dates back to 1991, when a cryptographically secured chain of blocks was described for the first time by S. Haber and W.S Stornetta. Seven years later, computer scientist N. Szabo worked on a project named "bit gold." Bit gold was about a decentralized digital currency that incorporated property chains powered by solving cryptographic puzzles. The project never reached maturity as it ran into a "double-spending problem," proving it unusable for its cause. [44]

In 2008, at the origin of a financial crisis, a project that would disrupt digital history and bring blockchain back to the surface came into existence. A pseudonymous creator named Satoshi Nakamoto presented a paper to a cryptography mailing list named "Bitcoin: A Peer-to-Peer Electronic Cash System." The paper laid out the schema for a peer-to-peer network that would support a "system for electronic transactions without relying on trust." Bitcoin incorporates a chain of digital signatures along with a proof-of-work system. The network avoids the double-spending issue thanks to the incentives provided to its peers. Soon after the scattering of the paper, the Bitcoin blockchain appeared with the genesis block mined on January third, 2009.

Getting away from cryptocurrencies, we will characterise a blockchain as a distributed database or a public ledger of records or transactions managed by peer-to-peer network based on a consensus protocol . The information is recorded in block of legitimate exchanges, and the record is shared and accessible to all hubs. The network works in a trustless manner which dispenses with the requirement for a central authority. The identity of participants is verified with the utilisation of cryptographic algorithms.

A characterising element of blockchain is unchanging nature. Contrasted with a regular database that upholds the CRUD (Create – Read – Update – Delete) activities, blockchain upholds just the CR part. Each block in the ledger cannot be erased after the verification from the majority of the participants. The history of transactions is permanent and unalterable. The previous trademark additionally empowers recognisability, which means any exchange can be followed back to its confirmed proprietor. Blockchains also own a part of their reputation to the reliability of the peer-to-peer network. The ledger is distributed and replicated in its entirety on every node. Therefore, if one or more of them are pushed offline, the integrity of the records is still preserved. All blockchain operations, from identity verification to the addition and verification of new blocks, are based on cryptography. Public key cryptography, hashing, and digital signatures are utilised. These basic components make blockchain an exceptionally protected kind of innovation.

3.2.1 Block structure

As the name discloses, blockchain consists of a sequence of blocks. How each

block is structured can reflect the integrity of this type of storage. Depending on the exact design of the block, the details it includes may vary. A generic blockchain structure can be depicted in Figure 3.6

Each block consists of a header and a portrayal of the exchanges it incorporates. The header can incorporate the current block hash, the hash of the previous block, and a timestamp. As examined in a past segment, hashing capacities save the information's honesty while holding the result to a proper length. We make a permanent chain by hashing the block and including its checksum into the following block. Any adjustment at any piece of a block would bring about a changed hash and, subsequently, an invalid chain.

3.2.2 Block creation

New blocks are produced repeatedly in the distributed network. Each partaking node can present its exchanges to a common pool, yet just the individuals who get into the following block are added to the chain. The demonstration of making another block in a blockchain network and getting it endorsed as the following block of the chain is called mining, he peer who participate in this process are known as miners.

Mining is part of blockhain's security process that creates a competitive environment where all miners are trying to validate data and check that everyone else who has mined a block has done so correctly. Consequently, approving blocks are frequently joined by some award from the network. Furthermore, it isn't extraordinary for members to remember an expense for their exchanges of incentives miners to prioritise them over others in constructing the next block. For the peer-to-peer network to work without trusted intermediaries, the process of maintaining the distributed ledger must generate enough incentive for attracting miners. Network reward and transaction fees serve as incentives, and blockchain protocol typically distribute them using a native token.

Depending on how the consensus is established on a given network, the mining process can differ significantly from model to model

3.2.3 How Does the blockchain Work

A blockchain practically fills in as a disseminated and secure database of exchange logs. In a Bitcoin network, to send some bitcoins to another client B, it will make a bitcoin exchange by client A. The exchange must be supported by miners before it gets submitted by the Bitcoin network. To start the mining system, the exchange is communicated to each node in the network. Those nodes that are miners will gather exchanges into a block, confirm exchanges in the block, verify transactions in the block, and broadcast the block and its verification using a consensus protocol (a.k.a., Proof of Work) to get

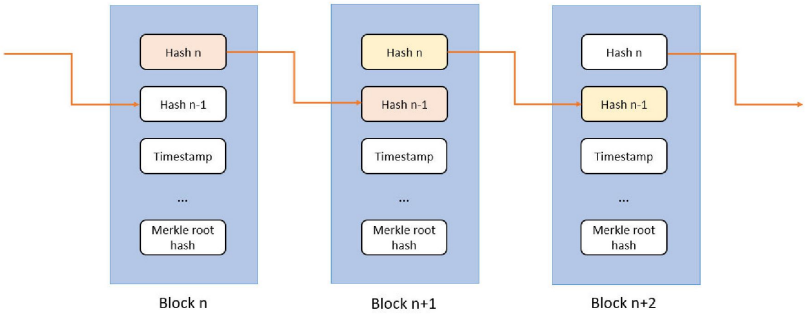

Figure 3.6: Basic blockchain structure

approval from the network. When other nodes verify that all transactions contained in the block are valid, the block can be added to the blockchain. Figure 3.6 gives a representation of this interaction. Just when the "block" containing the exchange is supported by different hubs and added to the blockchain, this bitcoin move from A to B will become finalised and legitimate. Three fundamental and significant abilities that are upheld by the blockchain execution in Bitcoin are: (1) the hash anchored capacity, (2) computerised mark, and (3) the responsibility agreement for adding another block to the globally chained storage. By an exquisite mix of a set-up of well known security methods, for example, Hash chain, Merkle tree, digital signature, with consensus mechanisms, the Bitcoin blockchain can forestall both the double spending issue of bitcoins and stop the review change of any exchange information in a square later the square has been effectively dedicated into the blockchain.

3.2.3.1 Hash Chained Storage

Hash pointer and Merkle tree are two fundamental building blocks for implementing the blockchain in Bitcoin using the hash chained storage.

Hash Pointer. Hash pointer is a hash of the information by cryptography, highlighting the area wherein the data is stored. In this way, a hash pointer can be utilised to actually take a look at whether or not the information has been altered. A block chain is coordinated utilising hash pointers to connect information obstructs together. With the hash pointer highlighting the archetype block, each block shows the location where the information of the archetype block is put away. Besides, the hash of the stored data can be publicly verified by users to prove that the stored data has not been tampered with.

Assuming that an enemy endeavours to change data in any block in the whole chain, to disguise the tampering, the foe needs to change the hash pointers of every past block. Eventually, the adversary has to stop tampering because he will not be able to falsify the data on the head of the chain, which is initially generated once the system has been built. We call this initial opening block in the chain the *genesis* block . At long last, the foe's altering will be revealed in light of the fact that by recording this single root hash pointer of *genesis* block, one can successfully cause the whole chain to have the property of tamper-resilience. Users are permitted to return to some special block and verify it from the beginning of the chain.

Merkle Tree A *Merkle* tree is characterised as a double inquiry tree with its tree nodes connected to each other utilising hash pointers. It is one more helpful information structure utilised for building a blockchain. Thusly, gathering these nodes into disjoint groups, to such an extent that each time two nodes at the lower level are assembled into one at the parent level, and for each pair of lower level nodes, the Merkle construction algorithm is creating a new data node, which contains the hash value of each. This process is repeated until reaching the root of the tree.

A *Merkle* tree has the capacity of keeping information from tampering by traversing down through the hash pointers to any node in the tree. In particular, when an adversary tries to tamper with data at a leaf node, it will cause a change in the hash value of its parent node; even if he continues to tamper with the upper node, he

needs to change all nodes on the path of the bottom to the top. One can without much of a stretch identify the information has been tampered, since the hash pointer of root node doesn't coordinate with the hash pointer that has been been stored.

An advantage of *Merkle* tree is that it can demonstrate viably and compactly the participation ofa data node by showing this data node and all of its ancestor nodes on its upward pathway to the root node. The participation of Merkle tree can be confirmed in a logarithmic time by processing hashes on the way and checking the hash esteem against the root.

3.2.4 Consensus models

The mechanisms that dictate how a decentralized, trustless blockchain network reaches an agreement are called consensus models. More often, this agreement refers to which block will be appended next to the blockchain. There are many models developed over the years like Proof of Work (PoW), the Proof of Stack(PoS), the Proof of Authority(PoA), and the Delegated Proof of Stack($DPoS$). Following, we are going to describe the first two as the most famous examples.

3.2.4.1 Proof of Work (PoW)

The Proof of Work(PoW) model is the most unmistakable illustration of agreement components. Hal Finney initially presented the Proof of Work model in 2004 for securing digital money through the idea of "reusable proof of work" utilising the SHA hashing algorithm. It depicts a cycle that permits a party to demonstrate that a significant amount of computational effort has been expended. Any spectator ought to have the option to check the demonstration of work with insignificant exertion.

A commonplace type of Proof of Work of Work incorporates the over and over hashing of a given info. Simultaneously, the digger appends an irregular nonce to it to compute a checksum with the mentioned attributes. For instance, with 'blockchain' a contribution to the SHA-256 hash work, we search for a ciphertext that should begin with five driving zeroes ('000000'). To tackle this riddle, it is basically impossible to get around it other than utilise brute force. A challenging excavator should begin by linking the underlying string with nothing, subsequently 'blockchain0', PC the checksum, and really look at its legitimacy. Assuming it doesn't follow the necessity, he should increase the nonce by one and make another endeavour.

blockchain0 →
bd4824d8ee63fc82392a6441444166d22ed84eaa6dab11d4923075975acab938

blockchain1 → db0b9c1cb5e9c680dfff7482f1a8efad0e786f41b6b89a758fb
26d9e223e0a10

blockchain2 →
3108428714518b4a2519e37ff8dce64d37c6ff9d8916cc6596391165db52c2b8

.....

blockchain1041 →
00007f73e777e83b01302b5fd5bc9905960c6398c7b24d0f2cc6a3e0c5cd3522

This interaction will require 1042 endeavors, yet in the end, a legitimate result will be created. In a real utilization of Proof of Work, the mentioned hash could incorporate at least 18 driving zeroes, which would take a lot of cycles. Nonetheless, it becomes apparent that paying little heed to the figuring power needed to deliver a substantial hash, the demonstration of confirming it stays as trifling as passing the showed answer for the hash capacity and really look at the outcome.[22]

Mining doesn't approve exchanges in frameworks that work utilising the Proof of Work as this errand is light computationally. It is utilised for building a solid responsibility against an assault all things considered. As more blocks are connected to the chain over the long haul, the amount of processing power needed to alter them turns out to be excessively awkward for an assailant to embrace. Therefore, the real condition of agreement turns out to be more strong as time elapses by. To converse or manufacture an exchange that happened a few blocks before, it would need to outperform the current development pace of the blockchain and recompute every one of the block added later the changed block. In this way, it is obvious that all Proof of Work blockchains stay secure as long there is sufficient figuring power supporting them in mining.

Consequently, Proof of Work receives lots of criticism as the hashing power it requires for a mainstream blockchain network to operate leads to significant energy consumption.

3.2.4.2 Proof of Stake (PoS)

The Proof of Stake (PoS)model is a promising consensus mechanism frequently recommended instead of Proof of Work. Miners in Proof of Stake are called validators. In this model, members set up their equilibrium of local blockchain tokens as icollateral. Consequently, they get approval over new blocks with respect to the sum they stake. Validators are picked haphazardly to make obstructs and are liable for approving and affirming blocks they don't make. The demonstration of approval blocks is regularly alluded to as validation.

Generally, validators get extra authority from responsibility for. As miners, they get compensated for making new blocks and bearing witness to blocks others made. A companion's stake likewise goes about as an extra motivating force to support great validator conduct. A temperamental validator can lose a piece of their stake for neglecting to approve impedes or even their whole stake assuming it endeavours to proceed as a troublemaker. Bearing witness to malevolent squares can likewise prompt loss of guarantee. [22]

As opposed to Proof of Work, validators don't require critical figuring ability to play out their obligations as they are arbitrarily chosen and not contending with one another. Proof of Stake reestablishes energy productivity and lifts any equipment constraints, permitting a more extensive crowd of friends to take part in agreement. Proof of Stake should prompt a more grounded resistance against centralisation in blockchain networks where token dispersion is assigned equitably. There are likewise

signs that Proof of Stake networks could manage scaling issues all the more proficiently utilising methods like sharding.

3.2.5 Blockchain types

Blockchains can be classified into different categories based on data accessibility, participation authorization, and core functionality with smart contract support

3.2.5.1 Accessibility

Based on data access, we can identify the following blockchain types:

- **Public:** In this kind, anybody can peruse and submit exchange giving a couple of keys. It is commonplace for the protocol and the code to be publicly released and kept up with by an open local area. The network works in a trustless manner Examples of public blockchain network are Bitcoin and Ethereum.

- **Private:** In this type, the blockchain is tightly controlled and established between trusted entities. A person or an organization operates the network. In this case, nobody can create new blocks or transactions on the network, verify the ledger history or participate in the consensus without explicit authorization from the owner or the central administrator. Blockchains in this category are based on solid trust to the owner and participants of his choosing.

- **Consortium or Federated:** In this sort, different gatherings of associations structure a consortium and are the just one permitted to submit exchanges and read from the shared ledger. This model endeavours to adjust public and private blockchains in trust and proficiency and is frequently considered to be mostly decentralised. Instances of consortium blockchains are Hyperledger and R3CEV.

- **Hybrid:** In this new kind have a place networks that join public, private, or consortium blockchains to streamline transactions. It is one more endeavour to join the smartest possible solution by keeping data private and evident by permitting access through a smart contract.

3.2.5.2 Participation

Based on the need for authorization to participate, blockchains can be divided in to the following categories:

- **Permissionless:** In this type, the ledger is shared with anyone to create new blocks without permission from a central authority. Participation in the network and blocks verification is allowed to everyone and is achieved using their computing resources.

- **Permissioned:** In this sort, interest is just allowed later express approval from an authority. There are instances of permissioned blockchains in which the admittance to exchange history or the making of new blocks can likewise be restricted to explicit members. Code conveyance can be open or shut source. In permissioned blockchains, agreement instruments can be nonexistent as the network works on trust between peers. This perspective makes networks of this kind high-speed and efficient. In the event of noxious action, the focal authority can boycott the agitator and reign in any unpermitted changes. In any case, permissioned blockchain are considered to forfeit decentralisation for execution.

- **Hybrid:** There is additionally a chance of a hybrid model among permissionless and permissioned blockchains. In this type, a node is taking part in a permissionless just as a permissioned blockchain to accomplish between blockchain correspondence. There may be instances of a solitary organization to be arranged to help both permissioned and permissionless operations.

3.2.5.3 Core functionality

Blockchains can be placed in the following categories based on the core functionality and the smart contract support:

- **Stateless:**In this sort, the blockchain network center is chain functionality and transaction optimisation. A new peer joins the networks can obtain the current state from the other nodes in the network. Every exchange can approved utilising self-included data. An illustration of a stateless blockchain network is Bitcoin. The networks supporting similar core functionality are also considered first-generation blockchains or blockchain 1.0.

- **Stateful**: In this kind, the network give smart contract backing and exchange registering capacities. Stateful blockchains acquaint rationale with improve and ensure their state and, they are regularly considered blockchain platforms. A blockchain of this class can uphold a wide assortment of uses by being programmed.

3.2.6 Blockchain Level Transaction Models

Blockchain is made and kept up with as a distributed ledger for online exchanges. There are two agent blockchain level exchange models: the upsent transaction outputs (UTXO) model, at first presented by blockchain and the account- based transaction model, presented by Ethereums. In this segment, we describe two transaction models and how their design difference may impact on the solution to the double spending problems.

3.2.6.1 The UTXO Model.

In Bitcoin and a significant number of its subsidiaries, a client stores the aggregate sum of her bitcoins as a rundown of "unspent" occasions of bitcoins that she has gotten however has not spent at this point. Utilising the UTXO model, the entire history of the bitcoin transaction in the system is recorded in a time series

3.3 An Introduction to Smart Contracts

3.3.1 What is smart Contract?

smart contracts can be viewed as an extraordinary development in blockchain innovation. In 1990^s, a smart contract was proposed as an electronic exchange protocol that executes the legally binding terms of an agreement. Authoritative statements that are implanted in smart contracts will be implemented automatically when a specific condition is satisfied (e.g., one party who breaches the contract will be punished automatically).

Blockchains are enabling smart contracts. Smart contracts are basically executed on top of blockchains. The endorsed legally binding provisos are changed over into executable PC programs. The coherent associations between legally binding provisions have been protected as sensible streams in programs (e.g., the if-else-if statement). The execution of each statement explanation is recorded as a permanent exchange put away in the blockchain. Smart contracts ensure fitting access control and contract enforcement.

Specifically, developers can allot consent for each function in the contract. When any condition is smart contract is satisfied, the triggered statement will automatically execute the corresponding function in a predictable manner. For instance, Alice and Bob settle on the penalty (as determined in the contract) will be automatically paid (deducted) from Bob's deposit.

A smart contract is a self-executing contract with the conditions of the understanding among purchaser and dealer being straightforwardly composed into lines of code. The code and the arrangements contained in that exist across a distributed, decentralised blockchain network. The code controls the execution, and exchanges are identifiable and irreversible.

Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism.

While blockchain technology has come to be thought of primarily as the foundation for bitcoin, it has evolved far beyond underpinning the virtual currency.

Smart contracts were first proposed in 1994 by Nick Szabo, an American computer scientist who invented a virtual currency called "Bit Gold" in 1998, fully 10 years before the invention of bitcoin. In fact, Szabo is often rumored to be the real

Satoshi Nakamoto, the anonymous inventor of bitcoin, which he has denied.

Szabo defined smart contracts are computerized transaction protocols that execute terms of a contract. He wanted to extend the functionality of electronic transaction methods, such as POS (point of sale), to the digital realm.

In his paper, Szabo also proposed the execution of contract for synthetic assets, such as derivatives and bonds. Szabo wrote: "These new securities are formed by combining securities (such as bonds) and derivaties (options and futures) in a wide variety of ways. Very complex term structures for payments can now be built into standardized contracts and traded with low transaction costs, due to computerized analysis of these complex term structures".

3.3.2 Life Cycle of Smart Contracts

The whole life cycle of smart contracts consists of four consecutive phases as illustrated in Figure 3.7:

1 Creation of smart contracts.

A few included gatherings initially haggle on the commitments, privileges and denials on contracts. Later numerous rounds of conversations and dealings, an understanding can reach. Attorneys or instructors will assist parties with drafting an underlying authoritative understanding. Programmers then, at that point, convert this understanding written in regular dialects into smart contract written in scripts including decisive dialects and rationale based guideline dialects. Like the improvement of program, the method of the brilliant agreement transformation is made out of plan, execution and approval (i.e.,. testing). It is worth focusing on that the formation of smart contracts is an iterative cycle including with numerous rounds of exchanges and emphasis. In the mean time, it id additionally engaged with numerous gatherings, for example, partners, legal advisors and computer programmers.

2 Development of smart contracts. The validated smart contracts can then be developed to platforms on top of Tangle. Contracts stored on the Tangle cannot be modified due to the immutability of blockchains. Any emendation requires the creation of a new contract. Once smart contracts are deployed on Tangle, all the parties can access the contracts through the Tangle. Moreover, digital assets of both involved parties in the smart contract are locked via freezing the corresponding digital wallets. For example, the coin transfers (either incoming or outgoing) on the wallets relevant to the contract are blocked. Meanwhile, the parties can be identified by their digital wallets.

3 Execution of smart contracts.

Later the advancement of smart contracts, the authoritative provisions have been observed and assessed. When the authoritative conditions reach (e.g., product reception), the legally binding systems (or functions) will be automatically executed. It is actually significant that a smart contracts comprises of various decisive articulations with legitimate associations. At the point when a condition is set off, the relating

articulation will be automatically executed, subsequently an exchange being executed and approved by miners in the Tangle. The serious exchanges and the refreshed states have been put away on the Tangle from thereafter.

4 Completion of smart contracts. Later a smart contract has been executed, new state of all elaborate parties are updates. Appropriately, the exchanges during the execution of the brilliant agreements just as the refreshed states are put away in Tangle. In the mean time, the digital assets host been moved starting with one gathering then onto the next party (e.g.,. cash move from the purchaser to the provider). Therefore, digital assets of involved parties have been opened. The smart contract then, at that point, has finished the entire life cycle.

It is worth focusing on that during development, execution and fruition of brilliant contract, an arrangement of exchanges has been executed (each relating to the assertion in the smart contract) and put away in the Tangle. In this way, this multitude of three stages need to compose information to the Tangle as displayed in the accompanying Figure 3.7

3.3.3 Properties

In an attempt to identify a smart contract accurately, the following characteristics can be extracted:

- **Smart contracts are deterministic.** This property means that if a contract distributed in various copies on several nodes receives a specific input, the same output shall be expected on all nodes. Therefore, a smart contract must not include any aspect of randomness or be affected by execution time.

- **Smart contracts are immutable.** This characteristic is the reason why smart contracts need to be executed on a blockchain. Blockchain enforces immutability. As soon as a contract is deployed on the public ledger, it cannot be modified. This aspect is mandatory for trust but introduces some challenges in case of bugs in the code.

- **Smart contracts are verifiable.** When a contract is being deployed on the blockchain, it receives a unique address. Any affected parties can use the address to locate the contract and verify its code and contents before proceeding with its execution.

- **Smart contracts are executed in a limited context.** This property means the contract can access their state, the context of the transaction that called them, and some information about the most recent blocks.

- **Smart contracts are decentralized.** A contract is executed on the local blockchain instance of a node, but as all instance share the same initial state, they deterministically produce the same final state. Thus the whole network operates as a single decentralized world computer.

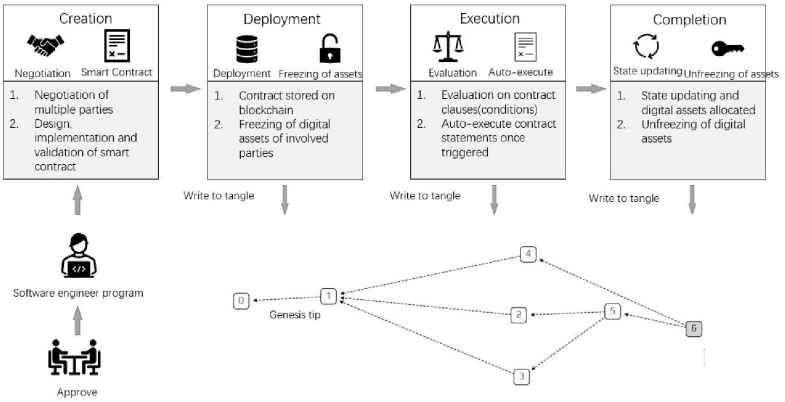

Figure 3.7: A life cycle of smart contracts

3.3.3.1 Oracles

The rules and conditions in a smart contract decide when an activity ought to be executed. At the point when a specific worth or blend of qualities has been accomplished, the smart contract is being executed. Its execution, in light of preconfigured algorithmic advances, plays out a progression of activities, bringing about another state or the discharge of an occasion. The consequence of the execution is written in the blockchain alongside the impact on the taking a participating parties.

There may be situations when the worth or the condition required is outside the organisation limits the contract is executed. The present circumstance prompts an issue as it drastically restricts the capacity of smart contracts to deal with real-world problems. An answer is given by entities called oracles. In the blockchain and smart contract context, an oracle is a specialist that finds and checks genuine occasions and communicates them as information in the blockchain. The essential obligations of oracles is to give such an information to smart contracts in a solid and confided in manner. Such information could be the worth of temperature, an effective cash exchange, the expansion of a stock cost, or the expansion rate.

There are many sorts of oracles, contingent upon how they are being used. Software oracles can deal with data that can be found online on the web. The wellsprings of the data might shift and could incorporate information or organisation websites. These oracles get the information from its internet based assets and forward them to any bought in smart contracts. Other smart contracts requires data straightforwardly from the actual world. For instance, a smart contracts could require values from a sensor situated at a specific area. The best test with equipment oracles is their capacity to report estimations without forfeiting information security. Oracles can likewise be grouped dependent on the course of the data. Inbound oracles can communicate data to smart contracts from outer sources. Also, outbound oracle can send data to outside world interfaces from smart contracts.

3.3.4 Decentralized applications

A decentralized application (dapp) is built on a decentralized network that combines a smart contract and a frontend user interface. In contrast to web applications, dapps do not rely on a regular backend server or a single computer to run, but they executed on the nodes of peer-to-peer network. The frontend of a dapp can be written in any language, just like an ordinary application, and make call to its backend.

As canvassed in the past section, a smart contract is a piece of programming that exists on the blockchain and executes exactly as programmed. When a contract is being sent to the network, it is absolutely impossible to transform it. Decentralised applications get their inclination from their backend parts. They are constrained by rationale composed into the contract and not by a person. This property is likewise a test as the smart contract fuelling an application should be planned carefully and tested thoroughly.

It should likewise be noticed that decentralized application existed before the presentation of blockchains. Instances of decentralized applications are BitTorrent

and Tor, intended to execute on distributed organizations that are not controlled by blockchain.

The contrast between a regular web application and a decentralized application with regards to smart contracts and blockchains can be portrayed schematically in [Figure 3]. A customer server application uses its frontend application to make HTTP solicitations to a backend API facilitated on brought together server. Simultaneously, a decentralised application utilises similar innovations for its customer to settle on decisions to the interfaces of smart contracts facilitated on a blockchain.

3.3.4.1 Characteristics

- The unique nature of decentralized applications inherits some very peculiar characteristics from its blockchain backend. Some of them are listen below:

- They are **decentralized**. These kinds of applications are independent and designed not to be controlled as a group. Their code is distributed to all peers of the network to be executed independently.

- They are **deterministic.** Dapps result at the same output given a specific input, irrespective of the environment they are executed. This characteristic is also a core property of smart contracts.

- They are **Turing-complete.** Based on compatibility theory, dapps are computationally universal. They are able to perform any action a modern programming language can perform.

- They are**isolated.** Decentralized applications are executed in a virtual environment running on the blockchain network, and therefore, they are fault-tolerant. In case of a bug or a severe malfunction, the application will not prevent the network from normal functioning.

3.3.4.2 Benefits

Like any traditional app, a dapp's purpose is to provide a solution to a problem. However, decentralized applications development introduces some notable advantages and benefits that typical applications do not.

Decentralised applications offer zero vacation. The smart contract fuelling a dapp is created on each hub taking part in the network. The network can generally react to customers hoping to connect with it. Denial-of-service attacks are not possible against a decentralised applications as the peer-to-peer network will mitigate any attempt.

Dapps also provide user privacy. Public key cryptography is used for identify verification against smart contracts and dapps. The users do not need to provide their real-world identity to interact with them.

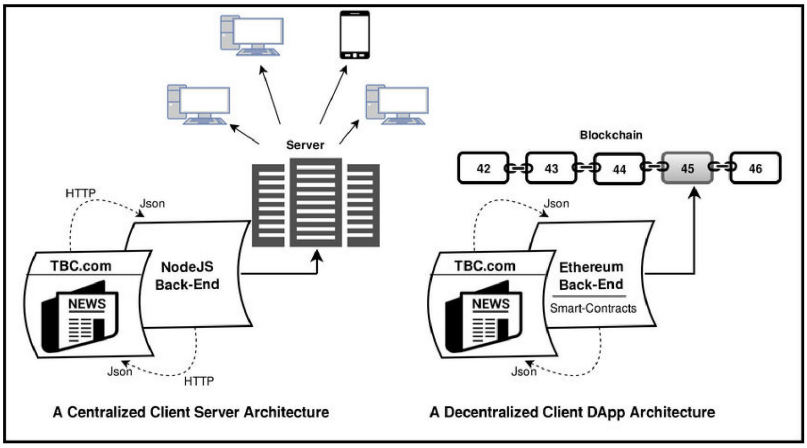

Figure 3.8: Dapp architecture comparision

A significant feature of decentralized applications is censorship resistance. No entity or central authority can block users from deploying or transacting with a dapp or reading data from the blockchain network.

Decentralized applications provide complete data integrity. Any piece of information stored on the blockchain is immutable and irrefutable preserved by cryptographic algorithms. No transactions can be erased, so data cannot be forged.

Trustless exchanges are one more remarkable attribute of dapps. There is no focal power or middle of the road needed for smart contract to be broke down and executed typically. This conduct may not be the situation on incorporated applications where the clients need to believe the server's proprietor that won't abuse records, mess with information, or even get hacked.

To wrap things up, decentralised applications are publicly released. The code executed in the backend in smart contracts is sent on the blockchain, so it is absolutely impossible to disguise malignant code. This angle permits a dapp to be checked on completely by the open-source local area to recognise bugs with or hidden pitfalls.

3.3.4.3 Implications

Of course, decentralized applications are not perfect and come with their own set of disadvantages and issues. Next, we will try to identify the most known implications dapps development often encounters.

Decentralised applications are difficult to keep up with. The code and the information of a dapp are sent on the blockchain, which makes them difficult to change. Blockchain engineers frequently face troubles to distribute updates to their applications or the information put away by them. Once a dapp is developed, even bugs and security risks are irreversible in the old version.

Decentralisation includes some major disadvantages. Decentralised applications manage genuine execution overhead. To profit from the upsides of blockchains, each node in the network should run and store each exchange. Decentralised agreement likewise incurs significant damage. Thus, there is extensive execution overhead for dapps, and scaling is one of the primary challenges they face today.

Dapps can face network congestion issues. The current model of blockchain platforms provides limited resources. If a dapps is not efficiently using network resources, the entire lattice can be hampered. If transactions are begin produced faster than the network;s capacity, the pool od unconfirmed transactions can grow significantly, leading to increased transaction fees and delays.

Decentralised applications lack on user-friendly experience. As it is currently at its beginning phases, the dapp ecosystem can be trying for the normal client. There is a heap of hardware needed to interface with the blockchain in a genuinely solid manner. Key stores are perhaps the greatest test current decentralised applications face. A lost key can turn the comparing identity unretrievable.

A frequently neglected part of decentralised programming improvement needs

to do concentrated executions. A dapp running on a blockchain doesn't authorise decentralisation on all parts of the advancement stack. Numerous dapps are killed by unified procedures consolidated as the second layer to the blockchain, regularly to further develop client experience. Outer administrations store keys or other private data, frontend clients served from brought together servers, or hybrid models that incorporate business logic to be executed on a server prior to being composed to the blockchain are normal models. All the above can take out the advantages of decentralised applications running on the blockchain.

Last, decentralized applications cannot handle KYC easily. Many centralized applications rely on user verification which can be easy when a single authority oversees the process. In the dapp model, there is no such entity, and for Know-Your-Customer, verification can be challenging in application domains that require it.

3.4 Potential solutions for smart contracts

Since the smart contract programs are conveyed in each node of the blockchain framework, the prerequisite of the precision of the smart contracts is incomparable. The smart contract is needed to be drained by vulnerabilities and programming errors prior to driving into the a great many blockchain nodes. There has been various exploration works led to address different security attacks over smart contracts.We discuss these research works for three main categories below: ***identifying semantic flaws, security check tools and formal verification.***

1. Identifying semantic flaws:

   Various examination studies have dissected these semantic imperfections and distinguished programming practices to moderate them. For instance, examined the weaknesses of Ethereum, which is well known in the business. The weaknesses were assembled into three classes as indicated by the level they are presented, as Solidity, EVM bytecode, or blockchain. The writers featured that they expect the non-Turing complete, human readable languages will resolve a portion of the issues distinguished.

   some significant insights from showing smart contract programming to under-graduate understudies in the University of Maryland.The authors exposed common errors in designing safe and secure smart contracts, and highlighted the importance of fixing these errors in programming.

   A few security designs which are appropriate to Solidity engineers in order to mitigate typical attack scenarios in Ethereum platform. The examples proclaimed included security of re-entrancy attacks, empowering mutexes and so on. The creators got ready for making an organised and instructive plan design language for Solidity.

2. Security check tools

   Further, different security check devices have additionally been support-

ive of presented to forestall semantic blemishes of smart contracts.

A few examinations have handled the ***re-entry attack***. The tool ReGuard, which are usable to identify re-entrency bugs in smart contracts. It is a fluffy based analyser which consequently recognises the re-entrency bugs in Ethereum smart contracts. ReGuard iteratively produces random different exchanges to test the vulnerability. Also, Jiang et al. [95] introduced ContractFuzzer, a far reaching fluffing system to distinguish seven kinds of vulnerability in Ethereum smart contracts. The creators distinguished not many significant kinds of assaults, for example, gas-less send and re-entrency vulnerability. The creators recognised the bogus negative rate enhanced when contrasting and different stages.

Other exploration works investigated ***Ethereum bytecode***. A security analysis frame-work for Ethereum smart contracts. It gives an investigation pipeline to the change of the low-level EVM bytecode into semantic logic relations. The assessment passed on that Vandal is quick and vigorous just as beating driving condition of-craftsmanship instruments with fruitful examination of 95 of each of the 141,000 unique contracts with a average runtime of 4.15 seconds. introduced Prosity, a decompiler which produces meaningful Solidity punctuation's from EVM bytecode. The decompiled contracts can perform with static and dynamic examination as required. later presented a complete small-step semantics of EVM bytecode and more formalised a fundamentally huge part of EVM utilising F*, which is a well known programming language utilised for comparable confirmation customised proof assistant. The creators likewise effectively approved it against true Ethereum test suite. The creators further characterised number of striking security properties for smart contracts. All the more as of late, introduced an open source dynamic execution frame- work named Manticore to analyse the binaries of Ethereum smart contracts. The framework provides analysis to find issues including logic bombs. The API gives adaptability to tweak the usage of system. This backings scalability up to bigger contracts. The creators tried the instrument with Oyente and noticed beating results and EtherTrust showed better accuracy on a benchmark rather than state-of- art solutions..

As Ethereum contracts consume gas, the gas cost on the smart contracts execution has likewise turned into a fundamental concern. GRECH et al. [100] characterised and distinguished the gas zeroed in on weaknesses found in the Ethereum smart contracts. The gas is the expense of a specific brilliant agreement execution on open Ethereum network and gas-focused vulnerabilities basically alluded to the codes with thorough execution to burn-through dispensed gas for the smart contracts. Moreover, the creators introduced MadMax, which are some static programming examination strategies usable to distinguish gas related weaknesses with altogether high certainty. The methodology remembered low-level examination for decompilation for declarative program analysis techniques for more significant level butt-centricysis which approved with 6.6 million contracts.

More broad purposes security check devices were additionally proposed. For instance, a far reaching investigation device that recognises security issues of Ethereum smart contacts. The creators evaluated the instrument on a gigantic

dataset of genuine contracts and yielded victories. They additionally expressed the capacity of advancement of the instrument in ongoing ways including improvement of language structure. Smartcheck has additionally been demonstrated to be more powerful in computerised security testing than different apparatuses . Another model, a security analyser for Ethereum smart contracts. It is adaptable, completely computerised and fit for demonstrating the agreement practices are protected or perilous comparing to a given property and tried with more than 18k contracts. The examination is a two ventured process which incorporates a representative investigation of contracts reliance chart to extricate exact semantic data and checking for the consistence violation designs. A representative execution tool named as Oyente to observe potential security bugs. The device hailed 8,833 contracts as vulnerable out of the 19,366 including the DAO bug which prompted a 60 million USD misfortune. Afterward, likewise introduced the structure Zeus, which can be used to check the rightness and approve the decency of smart contracts. They also defined the correctness as adherence to safe programming practices and fairness as the adherence to agreed upon higher-level business logic. The framework significantly outperforms Oyente with no bogus negatives in their informational index. Mavridou et al. presented FSolidM, which incorporated a fastidious semantics for planning contracts as limited state machines. The creators introduced an apparatus for making Finite State Machine (FSM) on an easy to understand graphical point of interaction which consequently generates Ethereum smart contracts. The creators additionally presented a bunch of configuration designs which can execute as modules and effectively conceivable to coordinate to improve security and functionality. Liu et al. proposed a semantic-mindful security evaluating procedure called S-gram for Ethereum. The creators consolidated N-gram language demonstrating just as lightweight static semantic naming and to learn factual controllers of agreement tokens and to catch undeniable level semantics like the stream affect-ability of an exchange. The creators showed that S-gram is usable to foresee likely weaknesses in distinguish sporadic symbolic groupings and conceivable to upgrade existing inside and out analysers.

3. Formal verification

   Formal check in everyday alludes to officially confirming the rightness of a PC program. Formal confirmation is significant with regards of smart contracts as smart contracts might hold monetary qualities and are regularly made open to everybody on a blockchain. A few examination studies have explored the conventional check of smart contracts, and applied proper confirmation on various stages during the organisation of brilliant agreements. For instance, Bhagavan et al . illustrated a structure to examine and check runtime wellbeing. While Abdellatif et al. and Nehai et al. proposed to check smart contracts in their execution environment. Albert et al. on the other hand proposed the EthIR framework to analyze Ethereum bytecode.

3.4.1 Privacy Issues and Solutions

Decentralisation is a core principle of the blockchain based smart contracts. The decentralisation in blockchain makes the exchange record and smart contracts straightforward

to all companions in the organisation as a component of safety. The straightforwardness isn't suggested for specific conditions. The in-built transparency is a significant privacy concern in blockchain based smart contracts.

1. Different Privacy Concerns

   Privacy is a more extensive space as far as the smart contracts [1]. Because of the appropriated idea of smart contracts, there are not many significant security concerns. These security concerns should be addressed to build the employability of smart contracts to the business .

   Transaction data privacy: In specific conditions, individuals in the network won't lean toward transparency since it will uncover some sensitive information such as trade secrets, pricing information. Despite the fact that the framework associated with blockchain, the necessary measures for privacy preservation should be integrated [25], [31] .

   Smart contract logic privacy: The smart contract publicly deployed in all nodes of the blockchain [10]. Because of certain necessities, the business rationale of the association needed to be joined in the blockchain. The business rationales might incorporate delicate data, for example, commissions, rewards. The noteworthy of such touchy data through the smart contracts will be a security worry of the associations.

   User Privacy: User privacy is highly concerned in some critical utilisation of blockchain based smart contracts. The clients join the smart contracts are needed to be private in specific conditions. For example, the arrangements like well being data frameworks don't like by the clients on the off chance that the individual character data being uncovered in the ledger. The security of user identity is additionally a critical worry in the execution of blockchain based smart contracts .

   Privacy in execution of smart contracts: The smart contracts are programs which execute on the computational framework . In blockchain, the smart contracts are executed on the nodes. The guidelines executed on the machine can be gotten to utilising various methodologies. For example, the password entered by a user needed to be stacked into the memory and can be seen in clear structure utilising memory dump instruments. These kind of lower level information thefts at the execution are viewed as privacy violations in specific conditions [5].

2. Potential solutions

   We below discuss and categorize various possible solutions on how to preserve smart contract privacy.

   Preserving privacy of transaction data: Ibáñez et al. [29] introduced significant insights on different parts of blockchain innovation remembering general information insurance guideline and its appropriateness for blockchain as an empowering agent for data protection. The authors talked about application of smart contracts on permissioned blockchains and permissionless blockchains

in relationship with fitting information regulators. The creators ordered the two sorts of arrangements in empowering consistence, as joining of various cryptographic capacities and private calculation plans without uncovering substance of exchanges and use of blockchains as decentralised confirmation machines.

Juels et al. [40] showed the rise of the criminal smart contracts which will work with to uncover the confidential information. The creators showed a couple of issues including robbery of cryptographic keys by criminal smart contracts. Their results featured making approaches and specialised protecting measures against criminal smart contracts to guarantee the smart contracts' beneficial objectives.

Kosba et. al [35] presents Hawk, a privacy preserving smart contracts, which disseminated the protection obstacle encountered in Bitcoin and Ethereum as a currency. The writers propose a framework, which empowers a non specialist programmer to write a privacy preserving smart contract.. Hawk guarantees ensures on-chain privacy, which cryptographically conceals the progression of cash and sum from public's view.

Protecting user privacy: Niya et al. [48] showed a plan and implementation of an exchanging application which used Ethereum smart contracts. The application is created with adaptability in mentioning client personality straight by the vendor and the purchaser. Lightweight blockchain is set up to work with information ex-change in gadget to-gadget channels.

Chatzopoulos et al. [16] proposed another design for the occasion based spatial crowd-sensing tasks in association with the blockchain and technology with user privacy preservation.The design uses smart contracts to permit swarm detecting specialist co-ops to present their solicitations, run cost ideal closeouts and handle payments.

Liang et al. [39] planned and executed ProvChain, which is a decentralised design for confided in cloud information provenance. Provchain gives critical security highlights, significant security features such as tamper-proof provenance and user privacy. The vitally functional stages are provenance information assortment, provenance data storage and provenance data validation which provides tamper-proof records to enable transparency and data accountability in the cloud.

Privacy in the logic

Al Bassam et al. [2] presents ChainSpace, which offers security amicable extensibility in the smart contracts stage. The stage offers higher versatility than the existing plat- form achieved through sharding across nodes using a novel distributed atomic commit protocol named as S-BAC. It also supports audibility and transparency.

Kalodner et al.[30] introduced Arbitrum, which is a cryptocurrency framework with smart contracts. Arbitrum's model is viable for private smart contracts which doesn't uncover the inward state to the verifiers who are engaged with the approval of exchanges in specific conditions. Arbitrum boosts the gath-

erings to concur off-chain on the VM's conduct which implies that the Arbitrum diggers simply needed to check advanced marks without uncovering the agreement to affirm that gatherings settled on VM's conduct.

Trusted execution environment (TEE): Trusted execution environment, for example, Intel SGX ensures classification and security during code execution.

Zhang et al. [25] introduced a validated information feed framework which is named as Town Crier. Town Crier gives an extension between smart contracts and existing sites which are regularly trusted for non-blockchain applications. The frontend and equipment backend joined with the arrangement which is empowered with privacy as required.

Cheng et al [17] introduced Ekiden, which joins blockchain with TEE. The creators utilised a novel architecture what isolates the agreement from execution and empowered classification protecting smart contracts in confided in execution environment. The creators intended to stretch out their work to empower secure multi-party calculation in future.

Yuan et al. [63] introduced ShadowEth, which is a system that use an equipment territory to guarantee the confidentiality of smart contracts in open blockchain like Ethereum. The framework likewise guarantees integrity and availability. The creators executed the model utilising Intel SGX on Ethereum network to investigate the security and vulnerability of the system.

Secure multi-party computation:

Benhamouda et al. [9] introduced a strategy for making Hyperledger Fabric blockchain stage viable with private information utilising secure multi-party calculation. The convention executed using Yao's millionaire issue [146] and oblivious transfer. The creators related an aide server, what isolates multi-party calculation into off-chain.

Zyskin et al. [67] introduced Enigma which is a computational model dependent on an exceptionally enhanced form of secure multi-party calculation named as Enigma which ensures an evident mystery sharing plans and guarantee confidentiality. The creators utilised a changed circulated hash table to hold secret-shared information to an outside blockchain as the regulator of the network to control the access and identity management. The private parts of the smart contracts run off-chain on Enigma stage and named as private contracts.

3.5 Technologies Involved

In this section, we will describe technologies implementing the fundamental theoretical presented up to this point. Many of these projects are pioneers in the domain of decentralized applications and still drive the current developments.

3.5.1 Ethereum

In 2013, an introductory technical paper was published by a young programmer, Vitalik Buterin. Highly influenced by Satoshi Nakamoto and the Bitcoin network, Buterin reinstates some of the concepts and combines them with the conception of Nick Szabo. He identifies that Bitcoin does support scripting capabilities that could be associated with smart contracts. However, Bitcoin scripting presents several significant limitations, as the absence of a Turing-complete language and the lack of state. Instead of inheriting Bitcoin limitations, Buterin proposes a new protocol named Ethereum for building decentralized applications.

Ethereum joins a blockchain with a built-in Turing-complete programming language permitting anybody to compose smart contracts and decentralised applications. The smart contract developer can make his own standards for possession, exchanges and state the board capacities.

Formal development of Ethereum started in early 2014 with the specification of putting executable smart contracts in the blockchain. The Ethereum Virtual Machine (EVM) is described in the Ethereum yellow paper as the computation engine which acts as a decentralized computer. EVM exists on every node participating in the Ethereum network. It is the part of Ethereum that runs execution and smart contract deployment. A contract is written in a high-level programming language and is compiled into a binary string named bytecode. EVM bytecode is the VM-level machine language that consists of opcodes, operation code instructions for the EVM to execute.

The first release of Ethereum was launched in 2015. The Ethereum Foundation, a non-profit organization, was created. To fund the development, Buterin and other co-founders launched a crowdsourcing campaign to sell participants Ether, the native Ethereum tokens. The public funding round was surprisingly successful, with the Ethereum Foundation raising several million to kickstart operations. Since then, the platform has evolved rapidly, with hundreds of developers involved worldwide.

3.5.1.1 Ether and Gas

Ethereum includes its native currency named Ether. As mentioned in the whitepaper, Ether:

serves the dual purpose of providing a primary liquidity layer to allow for efficient exchange between various types of digital assets and, more importantly , of providing a mechanism for paying transaction fees".

So, like Bitcoin, Ether is rewarded to miners as an incentive to protect the network security. Additionally, it serves as a mechanism for paying fees per transaction for anti-spam purposes. The smallest denomination of Ether is a Wei, named after the computer scientist Wei Dai. An Ether consists of 10^{18} Wei.

Transaction fees are part of the incentive mechanism in Ethereum as they are in Bitcoin. However, there is a difference in how they are represented and computed. In Bitcoin, the transaction fee is the difference in value between the input and the

output of a transaction. On the other hand, fees in Ethereum are not fixed as they are dependent on the code to be executed as part of the transaction. As stated in the whitepaper, a fee exists *"to prevent accidental or hostile infinite loops or other computational wastage in code."* They introduced a limit to the Turing-completeness nature of Ethereum's high-level programming language. While the language is Turing-complete, the developers is limited in the number of steps he can introduce in the implementation. As transactions are executed on every node in the blockchain network, a computational-heavy transaction burdens everyone. This way, fees also act as a measure against spam.

Every Ethereum transaction includes two fields named STARTGAS and GASPRICE. The STARTGAS is the dee the sender is willing to pay per unit of gas consumed. As a result, the total fee is calculated by multiplying the total gas the transaction used to execute with the GASPRICE. IT the total gas exceeds the amount of STARTGAS provided, then the transaction is reverted.

It is significant to get to the Ethereum figures out how to decouple the execution cost and the change in the worth of Ether with fiat currencies by providing a separate field for the gas price. This methodology is vital if, for instance, the cost of Ether increments dramatically on the cryptocurrency market. An exchange with a decent cost for every calculation could become restrictive. This issue is moderated using GASPRICE. In the past situation, regardless of whether how much gas devoured by a specific exchange stays steady, the gas cost can be diminished to permit the organisation to work at a sensible expense.

3.5.1.2 Accounts

The worldwide condition of the Ethereum is shut in account protests that can collaborate through messages with each other. Each record is recognised by a location of 20 bytes and keeps up with its own states. There are two sorts of records, externally owned accounts (EOA) and contract accounts. Accounts of the primary sort are constrained by private keys and have no code related with them. Accounts having a place with contracts are controlled by their contract code and have the relevant code associate with them.

The difference between the two account types are vital in understanding how Ethereum works. An externally owned account can send messages to other externally owned or contract accounts by creating and signing transactions using its private key. A message between two externally owned accounts is a transfer of value between the two. However, the message to a contract account activates the contract's code executing various actions defined by it. Those actions could be the transfer of tokens, the minting of new tokens, a write to storage, a calculation, the creation of a new account, to name a few. It should also be noted that contract accounts cannot initiate new transactions on their own. A contract account can only fire transactions in response to other transactions from either an externally owned account or a contract account. Consequently, any action that impacts the blockchain is always initiated by a transaction controlled by an externally owned account.

3.5.1.3 Token Systems

As Ethereum allows decentralized implementations, it introduces new notions like blockchain tokens. A token is a digital asset that exists on a blockchain but is not a built-in currency. It can be created and owned but also transferred to others, creating economic systems. Token systems are pretty easy to implement in Ethereum. [37]

It is essential to understand that a currency, or token system, fundamentally acts like a database with certain constraints. The database can execute a single operation simulating a transaction. A transaction is performed by subtracting X units from balance A and adding them to balance B. The constraints enforce that balance A had at least X units before the transaction, and A must approve the transaction. To implement a token system, all that is needed is to implement this logic into a smart contract.

One of the most renowned Ethereum tokens is ERC20, and it has emerged as the technical standard. ERC20 is used as an interface for all smart contracts to implement by providing a list of rules for fungible tokens. Other standards like the ERC721 represent ownership of non-fungible tokens (NFTs), where each token is unique. Various types of NFTs have become popular the recent years leading the frenzy of digital collectibles.

3.5.2 Solidity

Smart contracts are written in a higher-level language than the bytecode format of the Ethereum Virtual Machine (EVM). Solidity is the most popular smart contract language. As the Ethereum Foundation explains:

Solidity is a contract-oriented, high-level language for implementing smart contracts... It was influenced by C++, python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM)... Solidity is statically typed, supports inheritance, libraries and complex user-defined types, among other features.

3.5.2.1 Source file structure

Solidity source files can contain numerous contract definitions, import and pragma directives, struct, enum, function error, and constant definitions. As an example, let us inspect the following Solidity contract is Code 1:

```
1  // SPDX - License - Identifier: GPL -3.0
2
3  Pragma solidity >=0.4.16 <0.9.0;
4
5  contract SimpleStorage {
6     uint storedDate;
7
8     function set(uint x) public {
9         storedDate = x;
10    }
```

```
11
12      function get() public view returns (uint) {
13          return storedData;
14
15      }
16  }
```

Code 3 An example contract written in Solidity

At the first line, we encounter the software package data Exchange (SPDX) license identifier. In most circumstances, a contract code will be available as open-source. Every source file is encouraged to include a comment specifying the license under which it was released. The compiler does not validate this tag, but it is included in the contract bytecode metadata.

The pragma keyword found on the second line is used to indicate specific compiler features and checks. Every source file should be annotated with a version pragma to prevent incompatible compiler versions from introducing breaking changes to the code. For this example, the pragma directive specifies that the spurce code was written for Solidity version as early as 0.4.16. At the same time, it supports newer versions up to 0.9.0 but without including it [18]

The above example does not include one, but Solidity supports import statements

following the JavaScript syntax shown in Code 4:

```
1    \textnormal{import "./filename.sol";}
```

Code 4 An import statement

The above statement dictates that all global symbols for the source file specified by the import path should be included in the current file. This statement would be placed below the pragma version directive and above the contract definition.

3.5.2.2 Contract structure

In Solidity, contracts are very similar to classes in object-oriented programming languages. A Solidity contract consists of a collection of code and data. The contract code refers to its functions, while the contract data refers to its state. Every contract contains multiple declarations of state variables, functions, modifiers, events, errors, struct, and enum types as parts of its structure.

- **State Variables**

  State variables are variables that store their values permanently with the contract. The state variables can be modified by transactions executing code that modifies

their values. An example of a state variable is the *storedData* variable of the *SimpleStorage* contract above.

- **Function modifiers**

  Functions are very similar to other high-level programming languages. However, function modifiers are unique to Solidity. A function modifier looks like a function but uses the keyword *modifier* instead of the keyword *function*. The modifiers are called before the actual function and thus can change the function's behavior. They are prevalent to provide access control to functions which use should be limited according to specific conditions.

```
1   //SPDX - License - Identifier : GPL -3.0
2   pragma solidity >=0.4.22 <0.9.0;
3
4   contract Purchase {
5     address public seller;
6
7     modifier onlySeller() {   // Modifier
8        require(
9              msg.sender == seller,
10              "only seller can call this."
11          );
12        _;
13      }
14
15  function abort() public view onlySeller { // Modifier usage
16  // ...
17    }
18  }
```

Code 5 A function modifier

A function modifier with the name *onlySeller* can be found at line 10 in Code 6 above. The modifier validates whether the contract caller is the seller stored in the state variable using the *require* built-in function. If the validation fails, it will trigger an exception with the description passed as the second argument. The ; command will be reached otherwise, meaning the function execution will continue as if the modifier was never applied. The modifier is applied on function *abort()* at the line 15.

- **Events**

  Events are interfaces to EVM lumberjacks. Applications can buy in and tune in to occasions through the Ethereum customer. When radiated, they are put away as a feature of the transaction log alongside the arguments supplied. Assuming that a contention is named as *indexed*, it will be changed over into a point and become accessible. An occasion can have up to three points. The Log and its occasion information are not open from inside contracts, not even from the contract made. An illustration of a contract occasion assertion is situated at line 5 in Code 4 underneath. The event is terminated utilising the *emit* catchphrase inside the *bid()* work on line 9.

```
1   // SPDX - License - Identifier : GPL -3.0
2   pragma solidity >=0.4.21 <0.9.0;
3
4   contract SimpleAuction {
5        event HIghestBidIncreased(address indexed bidder, uint amount);  //
             Event
6
7        function bid()  public payable  {
8            emit  HighestBidIncreased(msg.sender, msg.value);   //
                 Triggering event
9        }
10  }
```

Code 6 A smart contract event

3.5.3 Current Research on Sharding

On the convention (Layer 1), Ethereum research is currently focused around a concept named sharding as a solution to the scalability challenge. Sharding is the methodology of isolating the blockchain into various equal chains with each chain keeping a security level around identical to the current single chain. This methodology will be introduced in more detail in Section , while the accompanying segment gives an outing into Proof of Stake for better comprehension of the foundation model for sharding .

3.5.4 Proof-of-stake

One of the major forthcoming changes in the Ethereum guide is a multi-stage progress from Proof-of-Work (PoW) to Proof-of-Stake (PoS). In the first stage2, the consensus mechanism will change into a hybrid PoW/PoS implementation called Casper. In Casper, the current blockchain is renamed to legacy chain. The legacy chain stays structurally unaltered and keeps on involving the PoW algorithm for agreement. Notwithstanding, the meaning of conclusion is introduction . Blocks on the legacy chain are presently deciphered as supportive of proposals for blocks on the new guide chain, which accomplishes consensus on a final and unchangeable history of blocks using the Casper FFG PoS algorithm. . A visual representation of the relationship between legacy chain and beacon chain is shown in Figure 3.9

This conclusion is accomplished utilizing supposed validators, which vote on new block dependent on protocol constraints. To turn into a validator, a network member is needed to store a huge negligible measure of ETH, the supposed stake. However long the validator takes part in the agreement and sticks to the convention imperatives, it will procure remunerates relatively to the stored stake. Infringement of convention rules prompts slicing (rebuffing the validator by consuming its whole stake) the store, while non-participation slowly drains the deposit to avoid deadlocking the network because of unavailable high-stake validators [[12]].

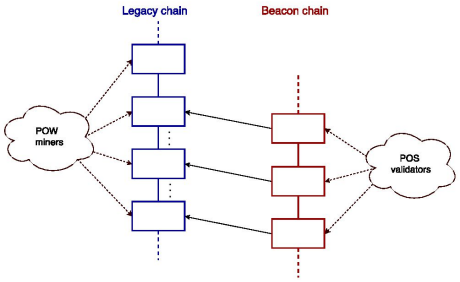

Figure 3.9: Casper FFG beacon chain as finalization overlay to the PoW legacy chain

3.6 Data Availability problem

Expecting the stateless client model, where clients are not needed to execute each exchange, keep the whole state away, or even download grouping bodies to confirm that all resemblances (and subsequently the present status) are legitimate, two impressive difficulties become obvious:

1. How might we guarantee that the whole information for each collation has at any point been published4? [8]
2. How to ensure that distributed information stays accessible as long as it is perhaps pertinent for future tasks?

A potential methodology is utilize employ fraud-proofs, where alleged fishermen (which could be any member or a particular job or if not limited gathering) , endless supply of inaccessible examinations, present a proof and guarantee a prize from the sliced store of the getting out of hand party. Clients could then stand by an adjustable time-frame depending on their risk tolerance if a fraud-proof for a specific state transition or version is published and in the case of absence of fraud-proofs, accept it as valid [11].

However, there is a significant problem with this approach resulting from the fact that unavailable data is not a uniquely

attributable fault. In the described scenario, it is impossible for an observer joining at T3 in Figure 3.4 to determine if the data publisher or the fisherman is acting maliciously.

This prompts an unthinkable choice in regards to the incentive model for fisherman in the portrayed situation as displayed in Figure 3.5:

1. A positive net compensation in that circumstance would permit fishermen to *forge* ETH through bogus cases.
2. An impartial net result prompts a Denial-of-Service (DoS) attack vector, where an attacker can constrain everyone to download each gathering body, adequately invalidating the advantages of stateless client and sharding.
3. Punishing the fisherman would lead to only altruistic fishermen reporting fraud-proofs and enables attackers to economically outlast these fishermen and consequently gain the ability to produce unavailable collations.

3.6.1 Current Approach

Bringing each of the moving parts together, we will depict the current methodology of settling the DAP in the sharding context of the Ethereum blockchain. As the theme is as yet in the condition of cutting edge research, this segment addresses to a greater degree an organized assortment of harsh thoughts rather than a formalized methodology.

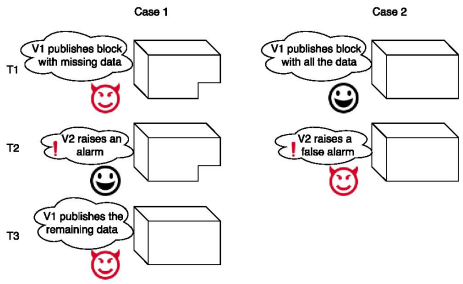

Figure 3.10: Scenario where unavailable data is not a uniquely attributed fault. Adapted from [8]

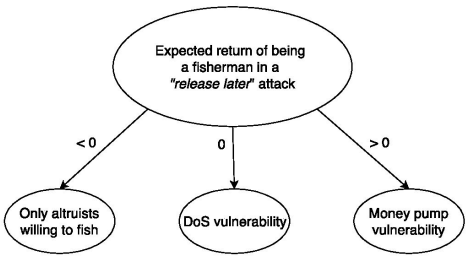

Figure 3.11: Impossible incentive model for fishermen Adapted from[8]

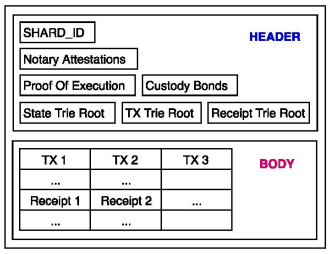

Figure 3.12: Example for a complete collation

3.6.2 Validation and Temporary Data Availability

To introduce an undeniable level outline of the validation process, we accept that assemblages are proposed in every shard while avoiding the particulars of the examination creation process.

Toward the start of every PoS *round*[5], a panel of validators is arbitrarily examined from the pool of Casper validators for every shard involving RANDAO as an arbitrariness source [52] . This panel is then answerable for downloading the whole body of the proposed grouping (a model is displayed in Figure 3.6) from the entity that created the proposal as well as a fixed number of parent collations, called the *windback*. Every member of the committee then generates a 1-bit custody proof following the process developed by Justin Drake [52]:

1. The validator generates a secret and publishes the hash of this secret.
2. The validator splits the data into chunks and XORs them with the secret and the hash of the secret.
3. The validator computes a Merkle Trie from the resulting chunks.
4. The validator publishes the hash of the data together with the least significant bit of the resulting Merkle Trie root.

The subsequent 1-bit guardianship bond is then transmission along with the validator's confirmation the gathering. Since a shard resemblance can allude to a beacon chain block involving it's hash in the assemblage header as well as the other way around, a single attestation is able to count as a validator's vote for a shard and on the beacon chain simultaneously.As soon as a sufficiently large committee has attested to a beacon chain block referencing a shard block, this reference becomes a cross-link, which tightly couples the state of the shard at the referenced block to the beacon chain and makes a reorganization of that shard impossible after the respective beacon chain block is finalized [53].

Subsequent to holding up a time of p_s , the validator is needed to produce another mystery and uncover the old mystery used to generate the care security. Now, every member in the network can confirm and challenge all care bonds published by that validator during the individual time span. Hence, the information accessibility is financially boosted until the challenge period p_c ,$p_c > p_s$ is over, which provides us with the ideal assurance for impermanent information availability [52].

3.7 The evolution of web

About 30 years ago, following the extended evolution of computer networking, a new technological notion started spreading to a mainstream extend. A technology that was destined to shape the identity of the modern world. By the early 1990s, the World Wide Web will have emerged, connecting millions of computers around the globe.

The network of networks would be relatively static during its first years. Plain pages consisting solely of text and images would pop out of the computer screens presenting their host's perspective to the world. The first era of the web, commonly referred to as Web 1.0, offered no interactivity to its users. It shared many similarities with the traditional media like radio and television broadcasts and consisted of providers feeding content to a handful of passive surfers. Most websites were built using static HTML pages stored in files and a few embedded styles.

As the audiences grew more extensive than before, many opportunities emerged, and the future seemed without boundaries. Every business sector, organization, and community sought online representation. Mass excitement followed by fear of missing out led many to defy business fundamentals and fueled a frenzy that was about to burst in early 2000. As it is widely knows, the dot-com bubble, a server financial crash came true with the technology-dominated NASDAQ index plummeting upto 77% in 2 years

The crisis caused billions to vanish into thin air but let the web progress into a steadier boom. The web technology stack evolved, upgraded servers, and faster internet connections spread the expansion. Many websites implemented features to allow their users to contribute and generate content. Browsing got easier and data transmission more viable. The interactive web, the second stage in the internet evolution, came into existence. Web 2.0, commonly referred to nowadays, was coined during a brainstorming conference between O'Reilly and MediaLive International in 2005 [50]

It is important to explain that Web 2.0 has no particular limits except for a center of standards and practices. During this stage, significant swifts occurred on how how software tends to be released, incorporating more agile and ongoing methodologies. Also, greater improvement organisations created some distance from the traditional software products to a more Software as a Service (SaaS) approach incorporating the value of data. Thus, has are currently the facilitators of client exercises on their sites rather than the sole suppliers of static substance. Remarkable instances of this are the online media networks and the media sharing stages, which comprise of the demonstrated prevailing entrances of the new web.

The evolution of web technologies, along with the 3G high-speed mobile networks of the late 2000s, led to the rise of the mobile web. New portable, lightweight, yet powerful devices, smartphones, and wearable hit the mass market, accessing the internet from almost anywhere. Based on the annual report of the ITU for 2014 [28], the number of mobile-broadband users reached 2.3 billion, while more than half of them originated from developing countries.

The massive generation of unstructured data needed to be permanently stored and efficiently retrieved to be analyzed forced new types of storage to emerge and pushed the NoSQL databases into the foreground. Defined as Big Data, the new web era introduced a mixed concept for data preserved in significant volumes. As a result, the necessity of a new discipline dedicated to data cleansing, preparation, and analysis was revealed. The field of Data Science materialized [24]

3.7.1 The web today

As the beginning of the third decade in the 21st century emerges, regular daily existence gets overwhelmed by the amazing accomplishments of the digital era. From work and finance to social connections and diversion, each viewpoint is impacted by the effect of information penetrating our lives through the web. In the new worldwide economy, digital growth is a significant indicator.

As illustrated in , there has been a steady raise in the number of users worldwide navigating the web over past 15 years. In addition, these figures are expected to skyrocket during the current year, as the COVID-19 pandemic forced even more people to stay indoors and, by extension, online. There are reports reveling a marked increase of 7.3% in the annual digital growth up to as of January 2021, approaching the significant percentage of 59.5% of the total world population.

As described in the previous section, the dot-com bubble burst in the early 2000_s. A critical reader would have expected the financial crisis to have fragmented the technological businesses ever since. In contracts, today's web is dominated by a handful of tech giants worth incredible amounts of dollars. As an indicative example, Apple Inc. was the first one trillion dollars company in history by august 2018 . The FAAMG (originated from the term FANG coined by Tim Cramer in 2013) is an acronym that stands for the five giant companies in the high-tech industry, namely Facebook, Amazon, Apple, Microsoft, and Google-parent Alphabet.

Based on the numbers presented, it is clear that the digital world holds the most significant role in modern civilization. It is now necessary to step back and reflect on how all these are supported.

3.7.2 Centralization

From the prior days of the web, the customer server model established the framework of the web's interconnectivity. The client is a PC that sends a solicitation, while the server is additionally a PC that reacts to it by sending data back to the customer. A significant perspective that should be featured is the focal place of the server in the client-server model. The server remains in the middle, serving demands at the same time from the customers encompassing it. There is no chance in this model for the customers to speak with one another straightforwardly, regardless of whether they are situated close to one another in the actual world.

The present customers comprise of billions of clients riding the web from their PCs or smartphones and the enormous number of IoT gadgets from sensors to brilliant vehicles or fridges. Likewise, the servers have additionally developed. From devoted PCs run by people to have their site during Web 1.0, the tremendous requirement for execution to effectively react to the inconceivable number of solicitations moved the servers from houses to colossal server farms. The image drawn is one of the billions of client served each second by enormous racks of servers co-situated in explicit spots. Typically, this makes a brought together model with servers assuming a basic part and is regularly depicted as the weak link.

3.7.3 The need for a decentralized web

The perils of centralization have been seen in the cases of significant incidents experienced on cloud providers comprising the backbone of the internet. On November 25^{th}2020, a flaw in the Kinesis Data Streams API of the Amazon Web Services (AWS) caused as outage affecting most North American data centers, knowing office popular third- party services. Most recently, another incident happened during the peak hours of December 14^{th} 2020, as an internal storage quota issue in Google Cloud infrastructure caused chaos in Europe. Major Google services like Gmail and YouTube were inaccessible for hours. However, most significantly, most of the third-party vendors utilizing Google for Single Sign-On (SSO) authentication were unable to operate.

Besides capacity and availability issues, centralization has also taken its toll most significantly on users' data, privacy, and security. All data from those tracking their activity to those comprising their digital identity are often collected and stored in corporate data centers. A data breach at Equifax in 2017 revealed the security risks associated with user details stored in a specific location. In 2013, many top-secret documents disclosed by ex-NSA contractor Edward Snowden revealed shocking evidence about mass surveillance projects by profit data-controlling entities on a global scale. At the same period, another data scandal broke out related to Facebook. Millions of personal profiles were collected and analyzed without user consent by the British political consulting firm Cambridge Analytica toe used in political advertising n

The models introduced above are only the pinnacle of the ice shelf in regards to the course the web has taken during its development during the 2010s. The master plan is drawn around the absence of command over data in an upset harmony between the clients and the stages, as portrayed in Figure 3. Bruce Sterling precisely illustrated this back in 2012 in his yearly discussion with Jon Lebkowsky on The WELL with regards to the condition of the world:

In 2012, it made less and less sense to talk about "the internet", "the PC business", "telephones", "Silicon Valley", or "the media", and much more sense to just study Google, Apple, Facebook, Amazon, and Microsoft. These big Five American vertically organized silos are re-making the world in their images. 3.13 .

This is without a doubt an extremely disturbing reality. A reality. A couple of huge stages guide the most traffic to online news sources and have huge impacts over what wellsprings of data general society devours every day. This could prompt server issues going from restriction by the public government to more subtle or unpremeditated inclining toward the substance clients see dependent on test unaudited proposal framework calculations incorporated into those united stages.

3.7.4 Web 3.0, the stateful web

Apparent from the previous paragraph, an alternative route to centralization appears mandatory. The solution may lay in a different approach in the networking model itself.

The peer-to-peer model is the same old thing. Promoted in the last part of the 1990s with the dispatch of Napster, a trailblazer document sharing programming, it

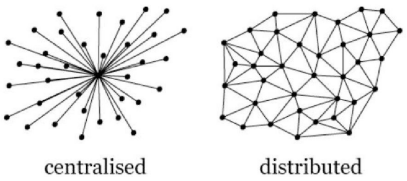

Figure 3.13: Centralized and Distributed Network

turned into the zeitgeist of web fans with the origination of the BitTorrent convention in the mid 2000s. Shared, or P2P as it is generally alluded to, is a dispersed application engineering dependent on similarly favoured interconnected friends that structure a peer-to-peer network With the shortfall of a focal server, the P2P model handles the issue at its root. Schematically portrayed in [Figure 3.13],the user can be connected to numerous peers, simultaneously optimising resource consumption and bandwidth based on location or other physical network factors. P2P networks have also been proved far more resilient. Any of the peers can go offline without disrupting the operation of the network as a whole. In contrast to the client-server model, the more users utilising a distributed resource in a P2P network, the more accessible that asset becomes to the remainder of the network, offering a solution to today's scalability issues.

The possibility of a decentralised web imparts the insight that a portion of the standards of P2P systems administration can be applied, aside from document sharing, to sites and web applications. This idea of web, additionally named Web 3.0, advances independence and uncovers another phase of movement on digital culture bit by bit grabbing hold. For the above vision to turn into a reality, one more innovation definitely known however sidelined for quite a long time must be consolidated.

As a thought additionally investigated in the following section, the blockchain was introduced generally to the general population with the arrival of the Bitcoin distributed cash in 2009 . This novel peer-to-peer network protocol can be used for peers to agree on the state of a distributed ledger of transactions. The immutability of blocks participating in a chain is achieved by cryptography. Blockchains reclassify the information designs of the web and establish the framework required for capacity and security for P2P appropriated applications in a trustless, decentralised way. This sort of decentralised applications, or dapps.

3.7.5 Web3.js

Web3.js is a collection of JavaScript libraries that allow interaction with the Ethereum ecosystem. As a convenience library, web3.js uses a provider to connect to a local or remote Ethereum node. The Ethereum provider JavaScript API is itself very simple and wraps Ethereum JSON-RPC formatted messages. This flow is depicted below in Figure 3.14

Web3 provides separate modules for different network functionality. The most popular is eth, which consists of tools for the Ethereum blockchain and smart contracts. CryptoCerts client uses web3.js for typical operations like fetching the active network identifier or the connected accounts and interacting with the CryptoCerts smart contract interface. Other modules include the *net* for interacting with network properties and protocol-specific modules like the *shh* focusing on Ethereum's peer-to-peer whisper protocol.

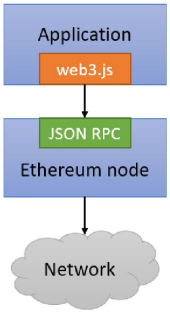

Figure 3.14: Web3.js to node communication

3.8 Tessera Private Transaction Manager

3.8.1 What is Tessera ?

Tessera is an open-source private transaction supervisor developed under the Apache 2.0 license and written in Java. The essential utilisation of Tessera is as the privacy manager for security empowered Ethereum clients like GoQuorum and Hyperledger Besu.

A privacy manager consists of two parts, each with their own liabilities:

1. Transaction Manager.
2. Enclave.

3.8.2 What can you do with Tessera?

1. Generates and maintain private/public key pairs.
2. manages and discovers all nodes in the network.
3. Provides an API for communication between Tessera nodes and an API for communicating with privacy- enabled Ethereum clients.

3.9 GoQuorum

3.9.1 What is Quorum?

Quorum is an enterprise blockchain platform. It is a fork of the public ethereum client 'geth' with a few convention level upgrades to help business needs. The basic role of the Quorum project is to foster an enterprise ethereum client which enables organisations to embrace and profit from blockchain innovation.

3.9.2 What can you do with Quorum?

Quorum has been presented in the expansive environment permissioned blockchains to meet the accompanying necessities:

1. Execute private transactions and smart contracts operations.
2. Adopt multiple consensus mechanism in a plug-in and play fashion.
3. Provide flexible and expressive network permissions management.

Quorum is worked later authority go execution of the Ethereum protocol (geth): The reasoning behind this decision is that, by limiting the progressions needed to Go- Ethereum, progression with future versions of the public Ethereum code base is effectively reachable. Every quorum nodes comprises of two fundamental administrations:

1. Quorum Client: It is the lengthy "geth" client, which is answerable for executing the Ethereum p2p protocol and the consensus algorithm.
2. Privacy Manager: It is a software module that empowers private transactions and private smart contract activities. It comprises of two parts they are Transaction Manager and Crypto Enclave.

3.10 Hyperledger Besu

3.10.1 What is Hyperledger Besu ?

Hyperledger Besu is an open source Ethereum client developed under the Apache 2.0 license and written in Java. It tends to be run on the Ethereum public network or on private permissioned networks, just as test network like Rinkeby, Ropsten, and Görli. Hyperledger Besu incorporates a few agreement calculations including PoW, PoA, and IBFT, and has complete permissioning plans planned explicitly for utilizes in a consortium environment.

3.10.2 What is an "Ethereum Client" ?

Hyperledger Besu is one of a few Ethereum clients. An Ethereum client is the software that carries out the Ethereum protocol. Ethereum client contain:

1. An execution environment for handling exchanges in the Ethereum blockchain.
2. Capacity for enduring data connected with transaction execution.
3. Peer - to - peer (P2P) network for communicating with the other Ethereum nodes on the network to synchronise state.
4. APIs for application designers to cooperate with the blockchain.

3.10.3 What are Hyperledger Besu's Features?

Hyperledger Besu carries out the Enterprise Ethereum Alliance (EEA) detail. The EEA detail was set up to make common points of interaction among the different open and close source projects inside Ethereum, to guarantee clients don't have seller lock-in, and to make standard interfaces of interaction for groups building applications. Besu carries out big enterprise highlights in arrangement with the EEA client detail.

Since it is viable with a public blockchain, enterprises can use Hyperledger Besu as a gate-way to associate with public Ethereum blockchain while getting their privacy necessities on a private blockchain. Besu is more appropriate to create financial applications that require security, high performance in private transaction processing since it is scalable, reliable and offers secured off-chain security. The Besu Ethereum client assists with sending smart contracts essentially with the help of tools, for example, Remix , Metamask and Web3js-eea customer library . Besu itself has a command Line Interface (CLI) and permits JSON RPC API calls to run, keep up with, debug and monitor nodes in a private Ethereum network. Besu upholds both Proof of Work (PoW) and Proof of Authority (PoA) consensus mechanism. The IBFT 2.0, Proof of Authority (PoA) consensus mechanism in Besu is fundamental to implement private transactions , furthermore it has prompt finality .The private transactions and blocks are validated by approved user accounts, who are called *validators* in IBFT Besu network. IBFT consensus protocol requires 2/3 or more validators to be on the web and alive to create blocks or execute a transaction.

3.10.4 What can you do with Besu ?

Besu incorporates an order line point of interaction and JSON-RPC API for running, maintaining, troubleshooting, and observing nodes in an Ethereum network. You can utilise the API by means of RPC over HTTP or through WebSockets. Besu likewise upholds Pub/Sub. The API upholds regular Ethereum functionalities, for example,

1. Ether mining.
2. Smart contract development.
3. Decentralized application (dapp) development.

3.10.5 What does Besu support?

The Besu client upholds common smart contract and dapp development, deployment and operational use functional use cases, utilizing tools, for example Truffle, Remix and web3j. The client supports common JSON-RPC API methods such as "eth", "net", "web3", "debug", and "miner".

Besu doesn't uphold key administration inside the client. You can utilize "EthSigner" with Besu to access your key store and sign exchanges .

4. Related Works

4.1 Ethereum

Ethereum is one of the greatest blockchain advances in 2022 that has a wide use case market in various industries like capital business sectors, decentralised finance, digital identity, government sector, insurance, medical care and production network the board. In view of the huge assortment of blockchains, picking the one that suites the reason best is a hard errand. Bitcoin is presumably the most known digital currency blockchain however it is focused around changing monetary financial systems. Ethereum rather has zeroed in on decentralisation of heavy computer frameworks and changing the current server-client foundation. The Enterprise Ethereum Alliance is the world's biggest business blockchain consortium with more than 450 individuals including Microsoft, JP Morgan, Accenture, ING, Intel, Cisco, and some more [14].

Ethereum is generally utilized through all kinds of industries from finance to production network the board and digital identity. For model Ethereum has brought numerous opportunities for supply chain industries in wording of discernibility, straightforwardness and tradeability. For character the board Ethereum brings the benefit of decentralised identity the executives and implanted encryption. Service providers can confirm the identity of a client which can then be conceded admittance by giving them a private key to recognise later once more.

Since there are many different options of blockchain technologies to be used in maritime environment, this thesis focuses on comparing them and analyzing which one could be the best possible solution. There are many ways of implementing a blockchain technology such as creating one's own network solution from the scratch and writing own smart contracts. This is however a lot of work and many easier solutions exist such as software clients of different blockchains. Usually they come with already existing consensus models and security methods or pluggable extensions for these actions. The "big three" between Ethereum Alliance, Tessera, Quorum and Hyperledger Besu can be chosen.

Options for which blockchain technology to use varies a lot whether Private, permissioned or public blockchain is needed. Private networks restrict adding new nodes to the network which limits scalability and flexibility for the environment architecture. Public networks instead are open for everyone which excludes the option of Proof-of-Authority consensus model and creates a vulnerable system for sensitive network communications.

4.1.1 Quorum

Consensys Quorum is a blockchain stage for big business arrangements. Some enormous endeavours in fields, for example, finance, exchanging and energy have entrusted Quorum with their blockchain needs and specifically associations like Microsoft and J.P. Morgan. Quorum was gained by J.P. Morgan in August 2020 with Consensys which implies consistent advancement and the board of Quorum administrations with on-demand support. Quorum is an open-source protocol layer that enterprises can create in-house applications on top of it. It bases on Ethereum and accompanies integratable product modules from Consensys [14] Open-source rendition of Quorum incorporates a private key manager EthSigner, a private transaction manager Tessera and two accessible client softwares Hyperledger Besu or GoQuorum. The endeavour stack of Quorum is more complex solution that accompanies extra modules like Codefi Assets, Codefi Markets and Codefi Payments which are not yet essential for the fairway project environment. With Quorum either permissioned or private network can be made which ensures information integrity, confidentiality and availability withfast transactions and operational transparency. Quorum gives broad aide on execution and additional documentation of Hyperledger Besu, Tessera and EthSigner. . On this thesis only Hyperledger Besu is covered but more information of GoQuorum can be found from [14].

4.2 Hyperledger Besu

Hyperledger Besu is a software client written in Java for Ethereum networks. It carries out Proof-of-Work and Proof-of-Authority consensus mechanisms. Hyperledger Besu goes about as the logging, checking, running and keeping up with clients and smart contracts and decentralised apps can be made and maintained with it. It can be run just as a command line interface or with JSON-RPC API

Engineering of Hyperledger Besu is partitioned in three sections: storage, Ethereum core and network. The capacity comprises of the genuine blockchain and afterward the historical backdrop of states and the current overall state that comprises of code and account states. Ethereum core handles the exchange functionalities by approving blocks/transactions and applying consensus. Network handles the fundamental peer-to-peer communications. Unique Tessera modules can be added to this architecture like better monitoring and encrypted storage's to further develop security.

Diverse consensus models can be applied to Hyperledger Besu client. There exists one Proof-of-Work consensus model Ethash and four choices for Proof-of Authority consensus: Clique, IBFT 2.0, Quorum IBFT 1.0 and QBFT. For private and permissioned networks PoA consensus depends on trust between members. They too have a lot quicker throughput than in PoW consensus based network. . Contrasts between PoA consensus models are likewise wide. Conclusion is an issue when numerous nodes attempt to do exchanges simultaneously. IBFT 2.0 and QBFT have immediate conclusion which implies each transaction block is added to the blockchain it is substantial. Clique anyway has a democratic system for exchanges and their legitimacy which can fork the blockchain quickly. This makes redesign in the blockchain periodically.

Validators of a transaction can shift. In Clique just a solitary validator can be picked however IBFT and QBFT need somewhere around four validators for an exchange to go through and be attached to the blockchain. It is presumed that all validators will satisfy their obligation and validate a transaction yet in the event that some don't react Clique is more fault-tolerate since just 50% of the validators need to sign the transaction.

To get security between nodes Hyperledger Besu has a private transaction manager Tessera. Every node that needs to impart transactions privately has their related Tessera node. On a side note private exchanges are as yet an early component on Clique or PoW consensus models. Every Tessera node and Hyperledger Besu hub have their own public and private keys and they utilise these to authenticate transactions that should be private. All correspondences use TLS. Private gatherings also have their nonce esteem. Privacy groups are distinguished by their ID and dealt with by Tessera. These groups can be flexible if their memberships are handled inside smart contract. Hyperledger Besu however collects the state of all privacy groups in addition to the state of the blockchain. All transactions going through Tessera are conveyed privately directly to the beneficiaries as opposed to sending them to transaction pool. Anyway a privacy marker transaction is shipped off the pool and later verified and added to the blockchain assuming nonce esteem stays right, in the event that not then the transaction won't be executed.

Since permissioned network is the most flexible but still secure solution Hyperledger Besu offers permissioning of nodes and accounts. Node permissioning means that only trusted nodes can access the network but account permissioning can restrict account's access to some nodes and it can be used to enforce identity requirements. Permissioning can be defined either locally or on-chain. Local permissioning happens inside one node and it can control who has the access. This is especially important in case of a threat or an attack because node owner can immediately protect it. On-chain permissioning affects the whole network and can take a while to coordinate the update through every node.

Transactions create logs that should be monitored but smart contract storage costs too much for extensive logs. Pluggable features can be added to Hyperledger Besu to improve monitoring and logging of network and node events. Plugins include commercial and enterprise versions such as Prometheus, Elastic stack, Quorum Hibernate and Splunk. All data can be visualized then in Grafana, Kibana or in Splunk's own graphical interface. There are two ways of changing logging: basic log level change and advanced format and output change. Log rotation can be changed to comply with regulations and standards.

4.3 Private Smart contracts

The privacy-enabled Hyperledger Besu network offers to deploy a *private smart contract* for a specific group or private participants. To execute a smart contract work on a private space, it needs to have precompiled s,art contract bytecode and a privacy marker transaction. The smart contract is compiled using solidity compiler *Solc* [45] also the executable EVM bytecode is put away in an Ethereum Besu node. It

is re-usable to execute any functions through private transactions with different privacy groups.A privacy marker transaction is created on the Besu network, and it has a pointer key (enclave) to direct an appropriate private transaction in Tessera node. . The address of the pre-compiled smart contract address is used as to attribute of the privacy marker transaction that will execute the smart contract function.

The *web3js-eea* is a JavaScript client library [18] that executes Hyperledger Besu's privacy-related operations such as create a privacy group, deploy private smart contracts, submit private transactions, view transaction status, and fnd/remove privacy groups. The web3js-eea method $eea_sendRawTransaction$ is used to submit private transaction with the attributes of *data, privateFrom, privacyGroupId*, and *privateKey* where the data is the binary code of pre-compiled private smart contract, *privateFrom* is the public key of Tessera node who sends the transaction, *privacyGroupId* d is the return value of generated privacy group id with a set of participants who involve in the transaction and the *privateKey* is the private key of Besu node who sends the transaction. The private transaction is passed from JSON-RPC endpoint to Besu's transaction handler that sends the transaction to the specified Tessera node. Then Tessera distributes the private transaction to all the Tessera nodes of users who are in the privacy group. The transaction data is stored in the recipient Tessera nodes with relevant transaction hash and privacy group ID. The transaction hash is returned from Tessera Besu's private transaction handler to create a privacy marker transaction to propagate for other users who are not in the privacy group.

4.4 Tessera

Tessera deals with blockchain's security for GoQuorum and Hyperledger Besu with transaction manager and enclave. Transaction manager handles dissemination of all private transactions, creates peer-to-peer network and moves transactions to and from ledger. More with regards to private transaction life-cycle can be perused from. All private and public key access and dealing with occurs in enclave. Enclave oversees keys and encryption and decryption processes that are required in transaction manager. Division of these two goes about as containerisation of sensitive data for security purposes that no leaks to unauthorised parties can happen. Logical separation always exists between enclave and transaction manager but they can be stored locally running in the same process.

Enclave can also be stored remotely in a HTTP process using RESTful endpoints and even in an additional environment for security purposes.

Tessera's enclave handles the main security parts of Besu's blockchain. It can fetch the identities, which is a node's public key, and distribute identities of forwarding nodes and overall handling all identity management. It also creates encryption of transactions and other payloads and decryption.

Privacy groups are managed by Tessera and there exists three types of groups: legacy, pantheon and resident. Legacy private group is created always when someone sends private data to another node. The group id for legacy group is created by hashing participants' keys. Pantheon is created before hand of transactions and the group id is

hashed similarly as legacy's but an additional seed is added to the group id. Resident group is very different since it is managed by Tessera configuration file and the id is just the name of the group.

4.4.1 Tessera's Transaction Manager:

In the life cycle of the private exchange payloads, the exchange chief is the essential issue of communication and distribution private payloads. It interacts with most different parts of the network and deals with the directional development of private data. Besu utilises a private transaction manager, Tessera, to carry out privacy. Each Besu node sending or getting private exchanges requires a related Tessera node.

Private exchanges pass from the Besu hub to the related Tessera node. The Tessera node encodes and straightforwardly disperses (that is, point-to-point) the private exchange to the Tessera nodes taking part in the transaction. it uses the Enclave for tasks including private key cryptography. This plan standard has been established to restrict admittance to an assailant that might have accessed a node or a transaction manager from reaching at private keys.Moreover, two significant network features of the transaction manager incorporate framing a peer-to-peer network of transaction managers to such an extent that peer/key data can be communicated, and interacting with the Enclave for cryptographic tasks even between nodes .

A network controlled by Tessera that empowers private payloads and dual transaction states has the potential for establishing an extraordinary off-chain testing environment explicitly for restrictive programming going about as payloads. Just the elaborate gatherings will approach the payload being tried, and the whole confirmation interaction can be computerised by a smart contract. When the check starts, a confined holder can test new information, distribute the outcomes to a blockchain, and afterward fall to pieces. This guarantees no unapproved admittance to payloads or any proprietary software. A board of validators keen on demonstrating the reproducibility and generalisability of a classifier can utilise the distributed outcomes from an off-chain test in making final recommendations.

Transaction manager where all transactions are validated, communication between different peers are established and exchanged.

1. Makes a peer to peer network with other exchange managers.
2. Delegates key administration and payload encryption/decryption to the enclave.
3. Stores and recovers saved information from the data set.
4. Conveys private exchange payloads for privacy-enabled Ethereum clients.

4.4.2 Tessera's Enclave

In network security terms, an Enclave is characterised as a solid registering resource that has no collaborations with the rest of the network, or some other frameworks. The

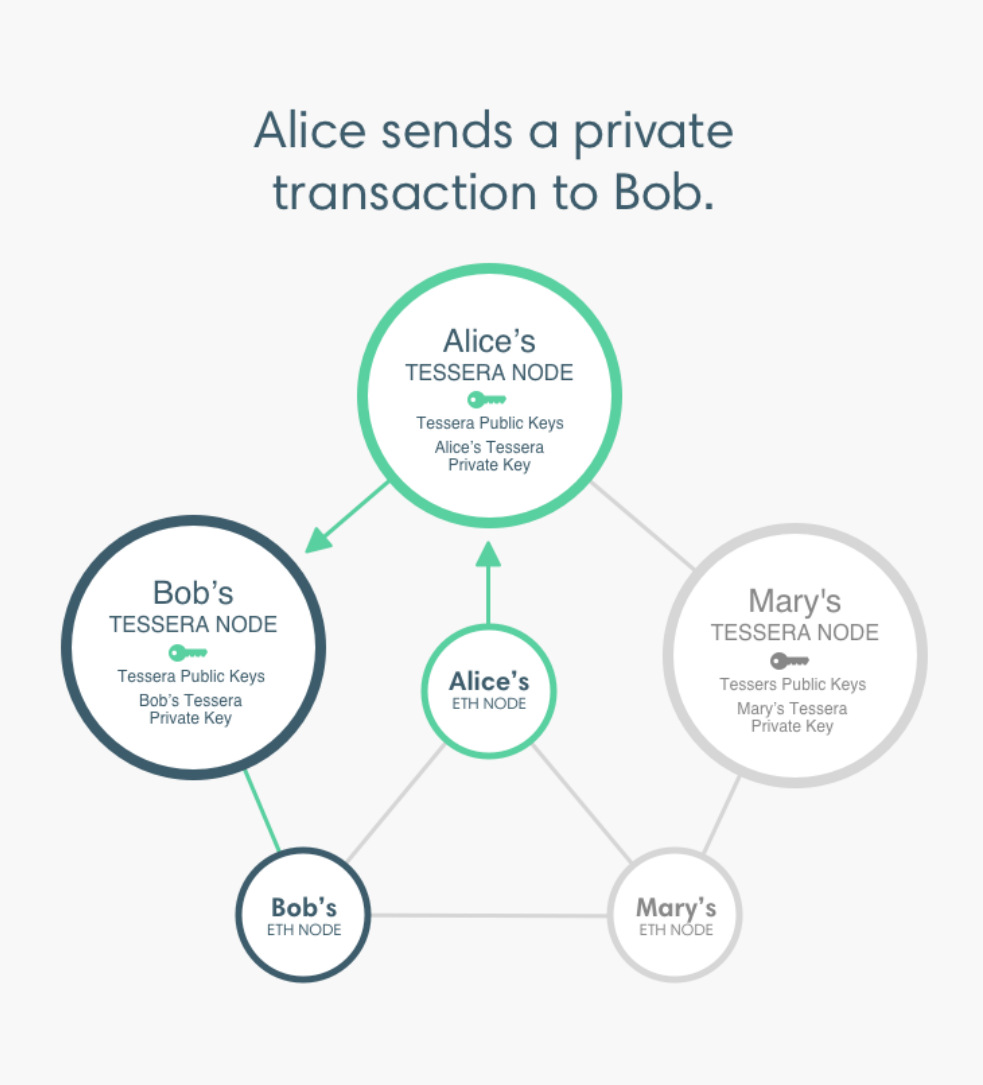

Figure 4.1: Private transaction

primary utilisation of an Enclave in Besu is to restrict network access and ensure data that exists within an Enclave from external malicious attacks.

Most distributed ledger protocols depend on cryptographic methods to keep up with transaction legitimacy, member verification, and the network state through a chain of cryptographically hashed headers. To accomplish real-time improvements for crypto-explicit activities and specialised segregation to make a crypto-space, a significant part of the cryptographic work including public/private key age and information encryption/decryption is assigned to the Enclave. It holds the private keys and capacities basically as a virtual hardware security module on the Besu network. The contention for a crypto-explicit space becomes applicable with regards to memory spills: Enclave works asynchronously with the transaction manager to assist with fortifying privacy by dealing with the encryption/decryption of tasks in a segregated way.

This empowers sensitive operations to be taken care of in a single container with layers of memory assurance, with next to no self evident potential for spillage into areas of execution memory that might be apparent to the rest of the network. The Enclave is intended to deal with a wide range of key management for Besu, just as supporting any activities needed by private payloads given that a specific node is taking part in the executing the payload on a virtual machine. In the most shortsighted functional model, every Enclave connects just with the transaction manager nearby to that node, yet there is support for additional complex interactions between transaction managers from various nodes including different Enclaves.

4.4.2.1 Enclave responsibilities

Data

The Tessera enclave handles:

1. Public and private key access.
2. Identities (public keys) of sending beneficiaries (*(alwaysSendTo)*
3. Default identity (public key) of appended nodes.

4.4.2.2 Actions

The The Tessera enclave performs the following actions on request:

1. Fetching the default identity (public key) for attached nodes.
2. Providing identities of forwarding recipients (public keys).
3. Returning all identities (public keys) managed by the enclave.
4. Encrypting a payload for given sender and recipients.
5. Encrypting raw payloads for given sender.

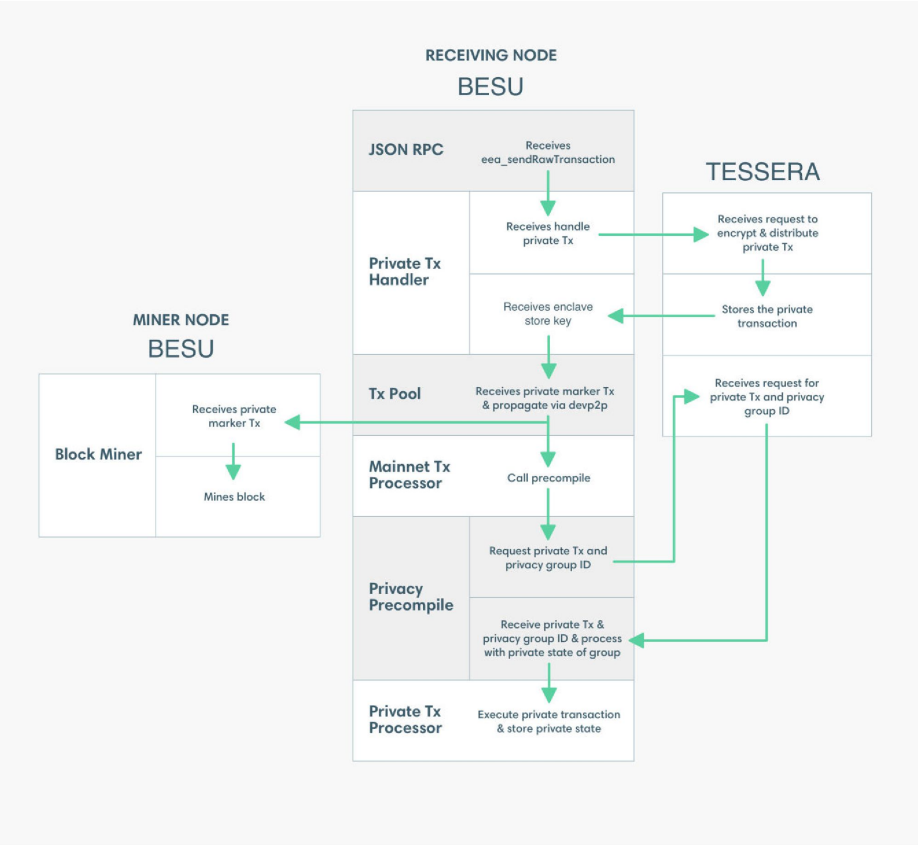

Figure 4.2: Life cycle of Private transaction

6. Decrypting payloads for a given recipient or sender.

7. Adding new recipients for existing payloads.

4.4.2.3 Enclave types

Tessera supports local and remote HTTP enclaves.

Local enclave:

The local enclave runs in the same process as the transaction manager. Logical separation still exists between the enclave and transaction manager.

Remote HTTP enclave:

The remote HTTP enclave serves RESTful endpoints over HTTP and runs as a separate process to the transaction manager. This provides a clear separation between the enclave and transaction manager.

For additional security, the remote HTTP Enclave can be deployed in a secure environment separate from the transaction manager.

4.5 Privacy groups

Privacy groups are managed by Tessera and there three types of groups: legacy, pantheon and resident. Legacy private group is created always when someone sends private data to another node. The group id for legacy group is created by hashing participants' keys. Pantheon is created before hand of transactions and the group id is hashed similarly as legacy's but an additional seed is added to the group id. Resident group is very different since it is managed by Tessera configuration file and the id is just name of the group.

4.5.1 Legacy Privacy Group

Tessera will consequently make a legacy group when a private transaction is sent to a list of recipient public keys utilizing *privateFor*.

while returning private transaction data to a Besu client (as a component of a / receive response), the legacy *privacyGroupId* will also be returned.

The legacy *privacyGroupId* is produced by hashing the rundown of recipient keys. For a given arrangement of recipients, there can be a single legacy privacy group.

4.5.2 Pantheon Privacy Group

Tessera upholds API strategies that empower the making of pantheon security groups from a viable blockchain client(for instance Besu).

Upon creation, the privacy group is appropriated to all individuals in front of transaction processing.

At the point when a private transaction contains a *privacyGroupId*, the exchange payload is distributed to everyone of the individuals from the privacy group.

The pantheon *privacyGroupId* is produced by hashing the rundown of recipient keys in addition to an irregular seed. This implies that for a given set of beneficiaries, there can be numerous pantheon privacy groups.

4.5.3 Resident Privacy Group

The idea of resident privacy group was acquainted with oblige GoQuorum multiple private states feature. A resident group contains a list of member keys that are locally managed, and those individuals will have a similar private state.

Key contrasts between a resident group and different kinds of privacy groups:

1. A resident group can only contain local keys and no remote key is permitted.
2. The name of the resident group will be used as the group identifier.
3. Resident group can be configured via Tessera configuration file, however, *multiplePartyStates* .

A number of validations will be obtained against the resident group configuration during Tessera startup (when *multiplePrivcyStates* feature is switched on):

1. All members of resident groups must be locally managed by the node.
2. A member cannot be configured to belong to more than 1 resident group at any time.
3. All keys configured must belong to a resident group and NO key can be left out.

4.6 Quorum Vs Hperledger Besu

Deciding on a Blockchain framework for your Usecase can be overpowering.

First, you need to decide: should it be – **public** or **permissioned** or a **private** second, if public then should it be **PoW, PoS** or prof of something and so on.

Definitions

firstly, let's see the definitions

Quorum is an enterprise blockchain stage. It is a fork of the public ethereum client *'geth'* with a few convention level improvements to help business needs. The basic role of the Quorum project is to foster an enterprise ethereum client which engages businesses to embrace and profit from blockchain innovation.

Hyperledger Besu, a Java-based Ethereum client previously known as Pantheon, is the first blockchain project submitted to Hyperledger that can work on a public blockchain. Besu addresses the developing interest of enterprises to construct both permissioned and public network use cases for their applications.

Essentially, Quorum and Hyperledger Besu have a similar vision of fostering a blockchain for enterprises on top of Ethereum. To make a blockchain for enterprises we significantly need three mystery fixings.

1. Permission layer
2. Private transaction
3. Enterprise architecture

4.6.1 Comparison

Permission layer

Assuming an enterprise is running a blockchain then they would with the exception of that every one of the characters in the chain are known and for a similar explanation, Quorum and Besu both have carried out a permission layer, which just permits the known hubs to join the network. For the two of them, the permission layer is fairly something similar and does its job impeccably.

Private Transaction Presently, Every association taking part in an enterprise-level blockchain will need to keep some sort of transaction concealed for different associations or users.

For this both, Quorum and Besu have carried out functionality of private transaction so while doing transactions you can simply state to make it private. To make a Transaction a user needs to make reference to the public key that will be simply ready to see the transaction then both Quorum and Besu has a different module that encrypts the transaction with those public keys and offers it with the comparing address utilising the network layers.

Presently, here comes the precarious part.

The problem of consistency- consider that user "A" has deployed a contract and made it private with user "B" and "c". The contract has an increment function which when called just increments the value of variable X (initial value = 0

by 1. Now " B" calls that increment function and makes this transaction private with only "C" so now the value of X for "B" & "c" is 1 while for "A" the value of X is still 0.

You should believe that this is a major issue and it totally challenges the fundamental idea of Private transactions in Besu and Quorum and Yes it is an intense issue. Also the central matter is that Quorum doesn't have any strong answers for this issue. There is a way utilizing which you can check the consistency on the organization layer however at that point this technique doesn't permits you to check the way utilizing which the worth arrived at the consistency.

In any case, luckily, Besu has a decent solution for this issue, they have carried out a thing call Privacy group utilising which you can to some degree tackle this consistency issue.

Enterprise Architecture

Each association has a hierarchy structure related with it, consequently, it is significant for any enterprise-specific platform to furnish the organisation with the Architecture which can uphold it.

Quorum has accomplished awesome work here and has carried out an architecture on top of Ethereum utilising smart contracts logic.

Besu has nothing like this except for it's not difficult to carry out the design structure Quorum into Besu. The engineer would simply have to send the contract from the Quorum in the Besu chain and the architecture would be fully operational.

Conclusion

In this way, obviously Besu is significantly better compared to Quorum and handles a portion of the issues that Quorum can't deal with, and for Bonus: Besu has a lot more small features previously carried out that ends up being truly useful for an enterprise- level blockchain.

4.7 Confidential execution of smart contracts

4.7.1 Ethereum Smart Contracts

Smart contracts are applications which are sent on the blockchain ledger and execute independently as a component of transaction validation. To convey a smart contract in Ethereum, a unique creation exchange is executed, which acquaints a contract with the blockchain. During this methodology the contract is relegated an exceptional location, in type of a 160-bit identifier, and its code is transferred to the blockchain. When effectively made, a smart contract comprises of a contract address, a contract balance, predefined executable code, and a state. Various parties can then communicate with a particular contract by sending contract-summoning transactions to a known contract address. These might trigger quite a few activities therefore, for example, reading and updating the contract state, cooperating and executing different contracts, or trans-

ferring value to other people. A contract conjuring transaction should incorporate the execution expense and may likewise incorporate a transfer of Ether from the guest to the contract. Moreover, it might likewise characterise input information for the conjuring of a capacity. When a transaction is acknowledged, all network members execute the contract code, considering the present status of the blockchain and the transaction information as input. The network then, at that point, settles on the result and the following condition of the contract by taking part in the consensus protocol Subsequently, on an applied level, Ethereum can be considered to be a transaction- based state machine, where its state is refreshed after each transaction.

4.7.2 Ethereum Programming Languages

Smart contracts in Ethereum are usually written in higher level languages and are then compiled to EVM bytecode. Such higher level languages are (Low-level Lisp-like Language) [58] Serpent (a Python-like language) [55], Viper (a python-like language) [59] , and Solidity (a JavaScript-like language) [6]. LLL and Serpent were developed in the early stages of the platform, while Viper is currently under development, and is intended to replace Serpent. The most prominent and widely adopted language is Solidity.

4.7.3 Solidity

Solidity is a high-level Turing-complete programming language with a JavaScript similar syntax, being statically typed, supporting inheritance and polymorphism, as well as libraries and complex user-defined types.

When using Solidity for contract development, contracts are structured similar to classes in object oriented programming languages. Contract code consists of variables and functions which read and modify these, like in traditional imperative programming.

```
1  pragma solidity ^0.4.17;
2   contract SimpleDeposit  {
3     mapping  (address =>  uint) balances;
4
5     event LogDepositMade(address from, uint amount);
6     modifier minAmount (uint amount){
7       require(msg,value >= amount);
8       _;
9      }
10
11     function SimpleDeposit ()  public payable{
12       balances [msg.sender]  += msg.value;
13       LogDepositMade(msg.sender, msg.value);
14     }
15
16     function getBalance ()  public view returns  (uint balance)  {
17       return balances [msg.sender];
18     }
19
20     function withdraw (uint amount)  public  {
21       if (balances [msg.sender] >=  amount)  {
22         balances [msg.sender]  -=   amount;
```

```
23          msg.sender.transfer (amount);
24          }
25      }
26  }
```

Listing 1. A simple contract where users can deposit some value and check their balance.

Listing 1 shows a straightforward contract written in Solidity in which clients can store some worth and actually look at their equilibrium. Prior to depicting the code in more detail, it is useful to give a few bits of knowledge about Solidity highlights like global variables, modifiers, and events.

Solidity characterises exceptional variables (msg, block, tx) that generally exist in the worldwide namespace and contain properties to get to data about a summon exchange and the blockchain. For instance, these variables permit the recovery of the origin address, how much Ether, and the information sent close by an invocation-transaction.

One more specific advantageous component in Solidity are alleged modifiers. Modifiers can be depicted as encased code units that advance capacities to alter their stream of code execution. This methodology follows a condition-orientated programming (COP) worldview, with the primary objective to eliminate contingent ways in function bodies. Modifiers can be utilised to effortlessly change the conduct of capacities and are applied by determining them in a whitespace-isolated rundown after the function name. The new capacity body is the modifiers body where $'\_'$ is supplanted by the first function body. An ordinary use case for modifiers is to actually look at specific conditions preceding executing the function.

A moreover significant and perfect element of Solidity are events. Events are dispatched signals that smart contracts can fire. Users and applications can tune in for those occasions on the blockchain absent a lot of cost and act likewise. Other than that, events may likewise fill logging needs. When called, they store their contentions in an transaction's log, a unique information structure in the blockchain that maps as far as possible up to the block level. These logs are related with the location of the contract and can be productively gotten to from outside the blockchain.

Given this short component depiction, we can now return and break down the code model. In the first place, the compiler form is characterised (line 1), then, at that point, the contract is characterised in which a state variable is declared (line 3), trailed by an event definition (line 5), a modifier definition (line 7), the constructor (line 12), and the genuine contract function (line 16 onwards). The condition of the contract is put away in a mapping called *balances* (which stores a relationship between a user address and a balance). The unique capacity *SimpleDeposit* is the constructor, which is run during the production of the contract and can't be called a while later. It sets the balance of the individual making the agreement (msg.sender)to how much Ether sent along the contract creation exchange (msg.value). The excess capacities really serve for collaboration and are called by users and contract the same. The deposit() function (line 16) controls the balances planning by adding the sum sent along the

transaction summon to the shippers balance, while using a modifier to fundamental guarantee that no less than 1 Ether is sent. The *withdraw()* function (line 25) controls the equilibrium mapping by deducting the mentioned sum to be removed from the senders balance and the *getBalance()* function (line 21) returns the genuine balance of the source by questioning the balances planning. In summary, this simple example shows the basic concepts of a smart contract coded in Solidity. Moreover, it illustrates the most powerful feature of smart contracts, which is the ability to manipulate a globally verifiable and universally consistent contract state (the balances mapping).

4.8 Development Aspects

1. ***Limits of Blockchain Technology :***

   To start with, it is vital to express that only one out of every odd application is foreordained to be run on a blockchain. There are numerous applications that needn't bother with a decentralised, permanent, append-only data log with transaction validation. Because of the innate attributes of blockchains, distributed ledger frameworks are not appropriate for an assortment of utilisation cases. For instance, calculation weighty applications are unreasonable to run on blockchains, due to the gathered calculation charges and the way that many kinds of calculations are unrealistic to execute on a stack-based virtual machine. One more impediment of blockchains is that they are not reasonable for putting away a lot of information. This certain impediment brings in the extensive redundancy from the large number of network nodes, holding a full duplicate of the distributed ledger. In any case, this can be overwhelmed by not putting away enormous information straightforwardly on the blockchain, however just a hash or other meta-data on the chain. With regards to information stockpiling it is likewise essential to understand that the information on the blockchain is noticeable to all network participants. This suggests that keeping sensitive data confidential, requires the requires of plaintext information by certain means. A further restriction of blockchains is their exhibition. They are right now not reasonable for applications which request a high-frequency or low latency execution of exchanges, in view of the extra work owed to the cryptography, consensus, and redundancy repetitiveness mechanical assembly of blockchain frameworks. Inside these cutoff points smart contracts ought to be utilised for applications that have something to acquire from being conveyed and freely certain and enforceable. As a rule, most applications that handle the transfer or registration of resources in a discernible manner are reasonable, for example land register, provenance documentation, or electronic democratic.

2. ***Coding Smart Contracts in Ethereum :***

   Contract advancement on the Ethereum blockchain requires an alternate designing methodology than most web and versatile engineers know about. Not at all like present day programming dialects, which support a wide scope of advantageous information types for capacity and control, the designer is answerable for the inward association and control of information at a more profound level. This suggests that the designer needs to address subtleties he may not be

utilised to manage. For instance, a designer would need to execute a strategy to concat or lowercase strings, which are errands engineers normally don't need to ponder in different dialects. Besides, the Ethereum stage and Solidity are continually advancing in a high speed and the designer is defied with a continuous change of stage highlights and the security scene, as new directions are added, and bugs and security risks are found. Designers need to consider that code that is composed today, will likely not order in a couple of months, or will essentially must be refactored.

3. ***Smart Contract Security :***

   An examination of existing shrewd agreements by Bartoletti and Pompianu [60] shows that the Bitcoin and Ethereum stage primarily focus around financial contracts. As such, most smart contract program code characterises how resources (cash) move. Along these lines, it is urgent that agreement execution is performed accurately. The immediate treatment of resources implies that imperfections are bound to be security significant and have more noteworthy direct monetary outcomes than bugs in commonplace applications. Episodes, similar to the worth overflow occurrence in Bitcoin [19] or the DAO hack [61] in Ethereum, caused a hard fork of the blockchain to nullify the malicious transactions. These occurrences show that security issues have been utilised for fake purposes savagely previously. A review of potential attacks on Ethereum contracts was distributed by Atzei and records 12 vulnerabilities that are doled out by setting to Solidity, the EVM, and blockchain quirks itself. A large number of these vulnerabilities can be tended to by following accepted procedures for composing secure smart contracts, which are dispersed all through the Ethereum community and distinctive Ethereum websites. Most accepted procedures fundamentally contain data about average entanglements to keep away from and the depiction of ideal plan and issue problem approaches.

4.9 Smart Contract Design Patterns :

A product design depicts a deliberation or conceptualisation of a concrete, complex, and repeating issue that software designers have looked with regards to genuine programming advancement projects and a successful solution they have carried out numerous occasions to determine this issue . Up until this point, just couple of endeavours have been made to gather and classify designs in an organised way [62] [62]. This segment gives an outline of average security configuration designs that are intrinsically incessant or viable with regards to smart contract coding. The introduced designs address run of the mill issues and weaknesses connected with smart contract execution. The examples depend on various sources, like audit of Solidity advancement documentation, on concentrating on Internet web journals and conversation gatherings about Ethereum, and the assessment of existing brilliant agreements. The source code of the introduced designs is accessible on github [64] . To represent the examples by and toward the finish of this segment records for each example a model agreement with distributed source code conveyed on the Ethereum mainnet.

Security Patterns :

Security is a group of patterns that introduce safety measures to mitigate damage and assure a reliable contract execution.

1. **Checks-Effects-Interaction :**

   CHECKS-EFFECTS-INTERACTION PATTERN

   Problem: At the point when a contract calls another contract, it surrenders control to that other contract. The called contract can then, at that point, thus, return the contract by which it was called and try to manipulate its state or hijack the control flow through malicious code.

   Solution: For a suggested useful code request, in which calls to outside contract are dependably the last advance, to decrease the assault surface of a contract being controlled by its own remotely called contract.

   The Checks-Effects-Interaction design is basic for coding capacities and portrays how capacity code ought to be organized to keep away from aftereffects and undesirable execution conduct. It characterizes a specific request of activities: First, check every one of the preconditions, then, at that point, make changes to the contract's state, lastly cooperate with different contract. Subsequently its name is "Checks-Effects-Interactions Pattern". As per this rule, connections with different contract ought to be, sooner rather than later, the absolute last advance in any capacity, as found in Listing 2. The reason being, that as soon as a contract interacts with another contract, including a transfer of Ether, it hands over the control to that other contract. This permits the called contract to execute possibly unsafe activities. For instance, an alleged re-entrancy attack, where the got back to contract calls the current contract, before the principal conjuring of the function containing the call, was done. This can prompt an undesirable execution conduct of functions, modifying the state variables to unexpected values or causing operations (e.g. sending of funds) to be performed multiple times. A model for a contract function, inclined to the portrayed attack scenario, is displayed in Listing 3. The re-entrancy attack is particularly harmful when utilising low level address.call, which advances generally remaining gas by default, giving the called contract more space for possibly malicious activities. Thusly, the utilisation of low level address.call ought to be stayed away from sooner rather than later. For sending reserves address. send() and address.transfer() ought to be liked, these capacities limit the danger of re-entrancy through restricted gas sending. While these strategies actually trigger code execution, the called contract is just given an allowance of 2,300 gas, which is presently a sufficient amount to log an event.

```
1     function auctionEnd() public {
2        // 1. checks
3        require (now >= autionEnd);
4        require (!ended);
5        //2. Effects
6        ended = true;
7        //3. Interaction
8        beneficiary.transfer(highestBid);
9        }
```

Listing 2. Applying the Checks-Effects-Interaction pattern within a function.

```
1     mapping (address => uint) balances;
2
3     function withdrawBalance () public  {
4
5       uint amount = balances[msg.sender];
6       require(msg.sender.call.value (amount)()); // caller's code is
            executed and can re-enter withdrawBalance again
7       balances[msg.sender] = 0;  //INSECURE - user's balance must be
            reset before the external call
8     }
```

Listing 3. An example of an insecure withdrawal function prone to a re-entrancy attack.

2. **Emergency Stop (Circuit Breaker)** :

EMERGENCY STOP (CIRCUIT BREAKER) PATTERN

Problem : Since a deployed contract is executed autonomously on the Ethereum network, there is no option to halt its execution in case of a major bug or security issue.

Solution : Incorporate an emergency stop functionality into the contract that can be triggered by an authenticated party to disable sensitive functions.

Dependably working contracts might contain bugs that are yet obscure, until uncovered by an adversary attack. One countermeasure and a speedy reaction to such assaults are crisis stops or circuit breakers. They stop the execution of a contract or its parts when certain conditions are met. A suggested situation would be, that once a bug is identified, all basic capacities would be ended, passing on just the likelihood to withdraw funds. A contract carrying out the depicted procedure is displayed in Listing 4. stop could be either given to a certain party, or handled through the implementation of a rule set.

```
1   pragma solidity ^0.4.17;
2   import  "../authorisation/ Ownership.sol";
3   contract EmergencyStop is Owned {
4     bool public contractStopped = false;
5
6     modifier haltInEmergency  {
7        if  (!contractStopped)  _;
8     }
9     modifier  enableInEmergency  {
10    if  (contractStopped)  _;
11    }
12    function  toggleContractStopped()  public  onlyOwner  {
13      contractStopped  = !contractStopped;
14    }
15    function  deposit ()  public payable haltInEmergency  {
16    // some code
17    }
18    function  withdraw()  public view enableInEmergency {
19    // some code
20    }
```

```
21      }
```

Listing 4. An emergency stop allows to disable or enable specific functions inside a contract in case of an emergency.

3. **Speed Bump :**

SPEED BUMP PATTERN

Problem : The simultaneous execution of sensitive tasks by a huge number of parties can bring about the downfall of a contract.

Solution: Prolong the completion of sensitive tasks to take steps against fraudulent activities.

Contract sensitive assignments are slowed down on an intentionally, malicious actions occur, the harm is limited and more opportunity to balance is accessible. A similar to true model would be a bank run, where an enormous number of clients pull out their stores all the while because of worries about the bank's dis-solubility. Banks normally check by postponing, halting, or restricting how much withdrawals. A model contract carrying out a withdrawal delay is displayed in Listing 5.

```
1    pragma solidity ^0.4.17;
2   contract SpeedBump {
3    struct Withdrawal {
4      uint amount;
5      uint requestedAt;
6    }
7    mapping (address => uint) private balances;
8    mapping (address => Withdrawal) private withdrawals;
9    uint constant WAIT_PERIOD = 7 days;
10   function deposit() public payable {
11     if(!(withdrawals[msg.sender].amount > 0))
12      balances[msg.sender] += msg.value;
13   }
14   function requestWithdrawal() public {
15     if (balances[msg.sender] > 0) {
16      uint amountToWithdraw = balances[msg.sender];
17      balances[msg.sender] = 0;
18      withdrawals[msg.sender] = Withdrawal({
19        amount: amountToWithdraw, requestedAt : now
20      });
21     }
22  }
23   function withdraw() public {
24   if(withdrawals[msg.sender].amount > 0 && now >
25          withdrawals[msg.sender].requestedAt + WAIT_PERIOD)
26           {
27      uint amount = withdrawals[msg.sender].amount;
28      withdrawals[msg.sender].amount = 0;
29      msg.sender.transfer(amount);
30    }
31  }
32 }
```

Listing 5. A contract that delays the withdrawal of funds deliberately.

4. **Rate Limit :**

RATE LIMIT PATTERN

Problem : A request rush on a certain task is not desired and can hinder the correct operational performance of a contract.

Solution : Regulate how often a task can be executed within a period of time.

A rate limit controls how frequently a function can be called consecutively inside a predefined time stretch. This methodology might be utilised for various reasons. A use situation for smart contracts might be established on employable contemplations, to control the impact of (collective) user conduct. As an illustration one would restrict the withdrawal execution pace of a contract to forestall a quick seepage of assets. Listing 6 exemplifies the application of this pattern.

```
1   pragma solidity ^0.4.17;
2  contract RateLimit {
3   uint enabledAt = now;
4     modifier enabledEvery(uint t) {
5      if (now >= enabledAt) {
6      enabledAt = now + t;
7    _;
8   }
9  }
10      function f() public enabledEvery(1 minutes) {
11       // some code
12    }
13  }
```

Listing 6. An example of a rate limit that avoids excessively retentive function execution.

5. **Mutex :**

MUTEX PATTERN :

Problem : Re-entrancy attacks can manipulate the state of a contract and hijack the control flow.

Solution : Utilize a mutex to hinder an external call from re-entering its caller function again.

A mutex (from mutual exclusion) is known as a synchronisation component in software engineering to limit simultaneous admittance to a resource. After re-entrancy assault situations arose, this example tracked down its application in smart contracts to shield against recursive capacity calls from external contracts. A model contract is portrayed beneath in Listing 7.

```
1   pragma solidity ^0.4.17;
2   contract Mutex {
3  bool locked;
4    modifier noReentrancy() {
5      require(!locked);
6      locked = true;
7      _;
8      locked = false;
9    }
10   / f is protected by a mutex, thus reentrant calls
11      // from within msg.sender.call cannot call f again
12    function f() noReentrancy public returns (uint) {
13      require(msg.sender.call());
14       return 1;
15   }
16  }
```

Listing 7. The application of a mutex pattern to avoid re-entrancy.

6. **Balance Limit :**

BALANCE LIMIT PATTERN :

Problem : There is always a risk that a contract gets compromised due to bugs in the code or yet unknown security issues within the contract platform.

Solution : Limit the maximum amount of funds at risk held within a contract.

It is for the most part smart to deal with how much cash in danger when coding smart contracts. This can be accomplished by restricting the all out balance held inside a contract. The example screens the contract equilibrium and rejects instalments sent along a capacity summon subsequent to surpassing a predefined standard, as found in Listing 8. It ought to be noticed that this methodology can't forestall the confirmation of coercively sent Ether, for example as recipient of a selfdestruct (address) call, or as recipient of a mining reward.

```
1  pragma solidity ^0.4.17;
2      contract LimitBalance {
3         uint256 public limit;
4         function LimitBalance(uint256 value) public {
5          limit = value;
6  }
7        modifier limitedPayable() {
8            require(this.balance <= limit);
9      _;
10 }
11       function deposit() public payable limitedPayable {
12         // some code
13   }
14 }
```

Listing 8. A contract limiting the total balance acquirable through payable function invocation.

According to Alharby and van Moorsel [64] current research on smart contracts is mainly focused on identifying and tackling smart contract issues and can be divided into four categories, namely coding, security, privacy and performance issues. The technology behind writing smart contracts in Ethereum is still in its infancy stage, therefore coding and security are among he most discussed issues. Unfortunately, a lot of research and practical knowledge is scattered throughout blog articles and grey literature, therefore information is often not very structured. Only relatively few papers focus on software patterns in blockchain technology respectively on design patterns in the Solidity language for the Ethereum ecosystem. Bartoletti and Pompianu conducted an empirical analysis of Solidity smart contracts and identified a list of nine common design patterns that are shared by studied contracts. These patterns broadly summarize the most frequent solutions to handle common usage scenarios. A paper by Zhang et al. [64] describes how the application of familiar software patterns can help to resolve design specific challenges. In particular, commonly known design patterns such as the Abstract Factory, Flyweight, Proxy, and Publisher-Subscriber pattern are applied to implement a blockchain-based healthcare application. The above mentioned papers do not contain security related design patterns, but show that design patterns are an interesting topic in smart contract coding.

4.10 Conclusion:

These concise prologue to Ethereum and Solidity and laid out six plan designs that address security issues when coding smart contracts in Solidity. By and large, the primary issue that these examples settle is the absence of execution control once a contract has been deployed, coming about structure the circulated execution climate given by Ethereum. This one-of-a-kind characteristic of Ethereum permits programs on the blockchain to be executed independently, yet additionally has downsides. These downsides show up in various structures, either as destructive callbacks, antagonistic conditions on how and when capacities are executed, or wildly high financial risks at stake. By applying the introduced designs, engineers can address these security issues and relieve common assault situations.

5. Conclusion

While great progress has been made in a short amount of time in building production-grade private blockchain systems with numerous features, much more work remains to be done to make these systems easy to use with appropriate tooling, standards and benchmarks to evaluate their performance. Mainstream database and distributed systems researchers and technologists need to get engaged in this space to ensure that their vast experiences are effectively leveraged to make these systems be even better along different dimensions.

With respect to various architectural features of the blockchain systems, the designers of these systems have made different choices. The rationale behind those choices are rarely well articulated and fully justified. Systematic comparisons of the different systems and their specific design choices remain to be performed. These systems could benefit from thorough and systematic investigations by researchers and deep technologists. Entrepreneurs could also find topics to work on to produce useful tools and methodologies to augment the existing blockchain systems.

One of the major challenges in the modern computer age is to protect the security and privacy of data. Moreover, organizations that take an integrated approach to handling and storing sensitive information are called upon to dedicate enhancing resources in order to secure appropriate information. The data provider gives access to a certain user(s) via Smart Contracts, where only authorized users are allowed to train their models on this piece of data.

IPFS-protocol-enabled decentralized storage nodes were integrated with our system to store data off-chain. Blockchain technology was used to enforce trust between participants and for immutable data transaction history and access log recording, trusted data provenance, and accountability features. Users can delegate their consent rights and audit them by tracking the event logs on the blockchain. Users are notified about any consent violation and can withdraw consent at any time. We designed and implemented a prototype of the proposed system, which is integrated with Hyperledger Besu blockchain to demonstrate the feasibility of our concept. “Privacy and scalability are the foremost reasons why organizations choose private blockchains over public blockchains,” based on research conducted by Hyperledger and the private transaction manager and their vulnerabilities with various research papers to provide appropriate information and challenges with the studies.

.

Bibliography and Sitography

[1] Muhammad Aitsam and Soamsiri Chantaraskul. Blockchain technology, technical challenges and countermeasures for illegal data insertion. *Engineering Journal*, 24 (1):65–72, 2020.

[2] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*, 2017.

[3] Merlinda Andoni, Valentin Robu, David Flynn, Simone Abram, Dale Geach, David Jenkins, Peter McCallum, and Andrew Peacock. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews*, 100:143–174, 2019. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2018.10.014. URL `https://www.sciencedirect.com/science/article/pii/S1364032118307184`.

[4] Christian Banks, Samuel Kim, Michael Neposchlan, Nicholas Velez, KJ Duncan, John James, A St Leger, and Daniel Hawthorne. Blockchain for power grids. In *2019 SoutheastCon*, pages 1–5. IEEE, 2019.

[5] Zijian Bao, Qinghao Wang, Wenbo Shi, Lei Wang, Hong Lei, and Bangdao Chen. When blockchain meets sgx: An overview, challenges, and open issues. *IEEE Access*, 2020.

[6] Massimo Bartoletti and Livio Pompianu. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International conference on financial cryptography and data security*, pages 494–509. Springer, 2017.

[7] Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, Hovav Shacham, and Scott Yilek. Hedged public-key encryption: How to protect against bad randomness. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 232–249. Springer, 2009.

[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[9] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM Journal of Research and Development*, 63(2/3):3–1, 2019.

[10] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

[11] V Buterin. A note on data availability and erasure coding. *github.com/ethereum/research/wiki/A-note-on-dataavailability-and-erasure-coding*, 2017.

[12] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[13] Yuanfeng Cai and Dan Zhu. Fraud detections for online businesses: a perspective from blockchain technology. *Financial Innovation*, 2(1):1–10, 2016.

[14] Wendy Marie Charles. Accelerating life sciences research with blockchain. In *Applications of Blockchain in Healthcare*, pages 221–252. Springer, 2021.

[15] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Arash Pourdamghani. Probabilistic smart contracts: Secure randomness on the blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 403–412. IEEE, 2019.

[16] Dimitris Chatzopoulos, Sujit Gujar, Boi Faltings, and Pan Hui. Privacy preserving and cost optimal mobile crowdsensing using smart contracts on blockchain. In *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 442–450. IEEE, 2018.

[17] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200. IEEE, 2019.

[18] Jitendra Chittoda. *Mastering Blockchain Programming with Solidity: Write production-ready smart contracts for Ethereum blockchain with Solidity.* Packt Publishing Ltd, 2019.

[19] Phil Daian. Analysis of the dao exploit. *Hacking, Distributed*, 6, 2016.

[20] Giuseppe Destefanis, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert Hierons. Smart contracts vulnerabilities: a call for blockchain software engineering? In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 19–25. IEEE, 2018.

[21] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[22] Tuyet Duong, Alexander Chepurnoy, Lei Fan, and Hong-Sheng Zhou. Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake. In *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*, pages 1–13, 2018.

[23] Dmitry Efanov and Pavel Roschin. The all-pervasiveness of the blockchain technology. *Procedia computer science*, 123:116–121, 2018.

[24] Aaron J Engelsrud. *Finding the data scientist within business intelligence practitioners.* PhD thesis, Capella University, 2017.

[25] Rajesh Gupta, Sudeep Tanwar, Fadi Al-Turjman, Prit Italiya, Ali Nauman, and Sung Won Kim. Smart contract privacy protection using ai in cyber-physical systems: Tools, techniques and challenges. *IEEE Access*, 8:24746–24772, 2020.

[26] Christopher G Harris. The risks and challenges of implementing ethereum smart contracts. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 104–107. IEEE, 2019.

[27] Huru Hasanova, Ui-jun Baek, Mu-gon Shin, Kyunghee Cho, and Myung-Sup Kim. A survey on blockchain cybersecurity vulnerabilities and possible countermeasures. *International Journal of Network Management*, 29(2):e2060, 2019.

[28] Masumi Hori, Seishi Ono, Shinzo Kobayashi, Kazutsuna Yamaji, Toshihiro Kita, and Tsuneo Yamada. Fusion of e-textbooks, learning management systems, and social networking sites: A mash-up development. In *International Conference on Genetic and Evolutionary Computing*, pages 377–386. Springer, 2015.

[29] Luis-Daniel Ibáñez, Kieron O'Hara, and Elena Simperl. On blockchains and the general data protection regulation. *University of Southampton*, 13, 2018.

[30] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1353–1370, 2018.

[31] Nikolaos Kapsoulis, Alexandros Psychas, Georgios Palaiokrassas, Achilleas Marinakis, Antonios Litke, and Theodora Varvarigou. Know your customer (kyc) implementation with smart contracts on a privacy-oriented decentralized architecture. *Future Internet*, 12(2):41, 2020.

[32] Shiho Kim and Ganesh Chandra Deka. *Advanced applications of blockchain technology*. Springer, 2020.

[33] Shinhae Kim and Sukyoung Ryu. Analysis of blockchain smart contracts: Techniques and insights. In *2020 IEEE Secure Development (SecDev)*, pages 65–73. IEEE, 2020.

[34] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858, 2016. doi: 10.1109/SP.2016.55.

[35] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[36] Keon Myung Lee and Ilkyeun Ra. Data privacy-preserving distributed knowledge discovery based on the blockchain. *Information Technology and Management*, 21 (4):191–204, 2020.

[37] Loïc Lesavre, Priam Varin, and Dylan Yaga. Blockchain networks: Token design and management overview. Technical report, National Institute of Standards and Technology, 2021.

[38] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. A survey on the security of blockchain systems. *Future Generation Computer Systems*, 107: 841–853, 2020.

[39] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477. IEEE, 2017.

[40] Anastasia Mavridou and Aron Laszka. Designing secure ethereum smart contracts: A finite state machine based approach. In *International Conference on Financial Cryptography and Data Security*, pages 523–540. Springer, 2018.

[41] Marco Mazzoni, Antonio Corradi, and Vincenzo Di Nicola. Performance evaluation of permissioned blockchains for financial applications: The consensys quorum case study. *Blockchain: Research and Applications*, page 100026, 2021.

[42] Andrew Miller, Zhicheng Cai, and Somesh Jha. Smart contracts and opportunities for formal methods. In *International Symposium on Leveraging Applications of Formal Methods*, pages 280–299. Springer, 2018.

[43] Tian Min and Wei Cai. A security case study for blockchain games. In *2019 IEEE Games, Entertainment, Media Conference (GEM)*, pages 1–8. IEEE, 2019.

[44] Marco Montanaro and Chiara Sarti. Cryptocurrencies as property: Solving the riddle. *De Lege Ferenda*, 4:1, 2021.

[45] Seyedehmahsa Moosavi. *Rethinking Certificate Authorities: Understanding and decentralizing domain validation*. PhD thesis, Concordia University, 2018.

[46] Joanna Moubarak, Eric Filiol, and Maroun Chamoun. On blockchain security and relevant attacks. In *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*, pages 1–6. IEEE, 2018.

[47] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[48] Sina Rafati Niya, Florian Shüpfer, Thomas Bocek, and Bürkhard Stiller. Setting up flexible and light weight trading with enhanced user privacy using smart contracts. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–2. IEEE, 2018.

[49] Pim Otte, Martijn de Vos, and Johan Pouwelse. Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 107:770–780, 2020.

[50] Tim O'reilly. What is web 2.0, 2005.

[51] Purathani Praitheeshan, Lei Pan, and Robin Doss. Private and trustworthy distributed lending model using hyperledger besu. *SN Computer Science*, 2(2):1–19, 2021.

[52] Daniel Sel, Kaiwen Zhang, and Hans-Arno Jacobsen. Towards solving the data availability problem for sharded ethereum. In *Proceedings of the 2Nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, pages 25–30, 2018.

[53] Amritraj Singh, Reza M Parizi, Meng Han, Ali Dehghantanha, Hadis Karimipour, and Kim-Kwang Raymond Choo. Public blockchains scalability: An examination of sharding and segregated witness., 2020.

[54] Amritraj Singh, Reza M Parizi, Qi Zhang, Kim-Kwang Raymond Choo, and Ali Dehghantanha. Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. *Computers & Security*, 88:101654, 2020.

[55] Sergei Tikhomirov. Ethereum: state of knowledge and research perspectives. In *International Symposium on Foundations and Practice of Security*, pages 206–221. Springer, 2017.

[56] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pages 9–16, 2018.

[57] Deepak K Tosh, Sachin Shetty, Xueping Liang, Charles A Kamhoua, Kevin A Kwiat, and Laurent Njilla. Security implications of blockchain cloud with analysis of block withholding attack. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 458–467. IEEE, 2017.

[58] Shuai Wang, Yong Yuan, Xiao Wang, Juanjuan Li, Rui Qin, and Fei-Yue Wang. An overview of smart contract: architecture, applications, and future trends. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 108–113. IEEE, 2018.

[59] Maximilian Wohrer and Uwe Zdun. Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 2–8. IEEE, 2018.

[60] Maximilian Wohrer and Uwe Zdun. Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 2–8. IEEE, 2018.

[61] Maximilian Wöhrer and Uwe Zdun. Design patterns for smart contracts in the ethereum ecosystem. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1513–1520. IEEE, 2018.

[62] Maximilian Wohrer and Uwe Zdun. Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 2–8. IEEE, 2018.

[63] Rui Yuan, Xia Yu-Bin, Hai-Bo Chen, Zang Bin-Yu, and Jan Xie. Shadoweth: Private smart contract on public blockchain. *Journal of Computer Science and Technology*, 33(3):542–556, 2018.

[64] Peng Zhang, Jules White, Douglas C Schmidt, and Gunther Lenz. Applying software patterns to address interoperability in blockchain-based healthcare apps. *arXiv preprint arXiv:1706.03700*, 2017.

[65] Shijie Zhang and Jong-Hyouk Lee. Double-spending with a sybil attack in the bitcoin decentralized network. *IEEE transactions on Industrial Informatics*, 15 (10):5715–5722, 2019.

[66] Shijie Zhang and Jong-Hyouk Lee. Double-spending with a sybil attack in the bitcoin decentralized network. *IEEE transactions on Industrial Informatics*, 15 (10):5715–5722, 2019.

[67] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.