Blockchain and Smart Contracts Technologies Comparison and Evaluation

By

Daniel Fox

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

24/09/2021

ACKNOWLEDGEMENTS

Throughout my project I have received a great deal of support and assistance. I would like to thank both my supervisors Dr Alexei Lisitsa and Dr Pamela Lins Bezerra, their expertise and guidance was invaluable in formulating the research topic and methodology. Special thanks to Dr Alexei Lisitsa for agreeing to initially be my primary supervisor for the proposed project.

DECLARATION

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

D.FOX

Abstract

The aim of this thesis is to compare blockchain and smart contract technologies using a quantitative analysis with an applied approach, primarily focusing on two areas of research within the industry, firstly blockchain developers, investigating blockchain platforms and the trade-offs they have to consider between decentralization, scalability and security within public and private blockchains, while also comparing consensus mechanisms which make the heart of the blockchain, the project compares three consensus mechanisms (Proof of Work, Proof of Stake and Proof of SpaceTime) by testing and forming research conclusions on the benefits and limitation of the three these include environmental effects, trade-offs and ease of use. Then secondly an investigation into smart contract languages focusing on features and their readability against Solidity, Go, Daml, outlining the benefits and limitations of each which formed interesting discussions on the aspects of readability of a smart contract and how beneficial the language would be for enterprises and consumers. The comparisons were made from research findings as well as writing up an Asset transfer contract in each of the languages. In the discussion section results will be discussed with my personal findings to introduced possible findings and considerations when looking to implement blockchain and smart contract technologies.

Contents

Table of Figures

1.0 Introduction

As blockchain development increases there is many problems within the space which need to be addressed, since the technology is still in its early stages there is possibly many underachieving problems. One of the main issues found with smart contracts is the wide variety of programming languages being used on different platforms, many platforms are using general purpose programming languages (GPLs) or domain specific languages(DSLs). The issue found with smart contracts is the fact that they are meant to be contracts, an agreement between two or more parties and while they are effective, they do have problems. Most likely the contract wants to be readable and understandable to non-developers, while most programming languages are difficult to read for a non-technical person.

The aim of this project is to analyses and evaluate both blockchain and smart contracts technologies and form a discussion on various features which would be useful to organisations looking to implement blockchain technology. The main goal is to perform experiments on smart contract programming languages and using both a permissionless and permissioned blockchain platforms to help formulate decisions on what features found that could be important and beneficial to organisations by taking a practical approach to gain personal understanding, experience and to further back up the academic research being carried out.

Multiple smart contract languages will be used to personally test for a series of features which could be beneficial for organisations looking at implementing smart contracts. Another area is the blockchain platforms making a comparison between platforms and their consensus algorithms to figure out what is the most efficient and beneficial to use. Research was carried out alongside the testing and analysing blockchain and smart contract technologies to compare finding, features and benefit types of blockchain platforms and smart contracts offer

1.1 Summary of Project

The main goal of the thesis is to form a comparative analysis on blockchain technology and smart contract programming languages. Focusing on two main areas:

A comparison between two widespread Consensus mechanisms Proof of Work and Proof of Stake and a newer Proof of SpaceTime Consensus mechanism, the comparisons will be discussed from experience running my own node and comparing discoveries against research found. Comparing trade-offs, security risks as well as environmental factors, and a brief discussion on my experience running each validator.

The second area of the project aims at smart contract developers, primarily discussing programming languages features and comparisons between general purpose languages and domain specific languages, these languages include: Solidity on Ethereum , Go on Chaincode, Daml a multi-platform business logic language. The purpose of this part is to form a discussion on the importance of making smart contracts and the potential problems with smart contract designs in a business-like environment, aiming the discussion at readability concerns, ease of use (syntax, documentation) and popularity as well as discussion the benefits of domain specific languages and general-purpose languages.

1.2 Aims and Objectives

The aim of this project is to research and design blockchain and smart contract technologies and formulating a comparative analysis based on personal findings from designs and tests and comparing them against areas of research.

The project looks at two areas within blockchain technology:

- Blockchain and Consensus Mechanisms – A brief comparison of public and private blockchains discussing trade-offs and a deeper investigation into different consensus mechanisms which securely verify transactions on the network.
- Smart Contracts - The technologies built on top of blockchain, programable contracts which allow developers to create applications which can be automated and executed on the blockchain.

1.3 Research Questions

The questions set throughout the research as listed below:

Research Question 1 : Blockchain Trade-offs against Decentralization, Scalability and Security describing the benefits of each and how they are used in Hyperledger and Ethereum a private vs public comparison.

Research Question 2 : Consensus mechanism comparisons , known to be the heart of the blockchain providing the most effect to the blockchain trade-offs. Testing running nodes on Proof of Stake, Proof of Work and Proof of SpaceTime and comparing my findings with other research to discuss benefits and trade-offs, limitations , security concerns , ease of participation and environmental effects.

Research Question 3 : Comparing Solidity , Go and Daml smart contract languages and comparing the benefits and limitations of each relating to personal findings from developing smart contracts in all languages and comparing with research areas. The discussion will focus on ease of implementation and measuring metrics like documentation , resources, syntax, security, and readability.

2.0 Background

2.1 Blockchains and Distributed Ledgers

A blockchain is a type of distributed leger, distributed ledgers are use independent computers typically referred to as nodes to record, share and sync records of transactions evenly across each node in the network, instead of synching to one central database. [1].

Participating in the network as a node requires contribution to the ledger, as the data is distributed across the entire network its important to make sure all nodes agree or at least the majority, changes to one node are propagated by all other nodes. The common truth agreed upon nodes is called consensus, these are protocols used to help ensure the security of the blockchain [2].

Distributed ledgers can be categorized into unpermissioned and permissionless. Unpermissioned ledgers have no single owner, and they allow write access by any person, they are public which allow unrestricted read access by anyone. Thus, enabling anyone to initiate and receive data through public unpermissioned blockchains, it also allows anyone to be a node operator on the network and participate in the consensus protocol. A good example of a unpermissioned ledger is Ethereum which will be covered within this project.

Permissioned ledgers can have one to many owners in the network however participants are selected to join by some central authority who can restrict and limit access to the ledger. An example of this is Hyperledger Fabric which will be compared alongside Ethereum within the report.

The blockchain itself is a type of data structure that is used as an application in many distributed ledger technologies. The name is blockchain is formed from how data is collected, new additions to the ledger are initiated by one of the nodes, who create a new block of data such as transaction records, the information about the block is shared across the network of ledgers, the data will be encrypted so transaction details aren't made public, all nodes in the network will then validate the block according to the consensus mechanism. After validation, all participants update the block to the ledger and through this each ledger has an identical copy of the network on their system [3].

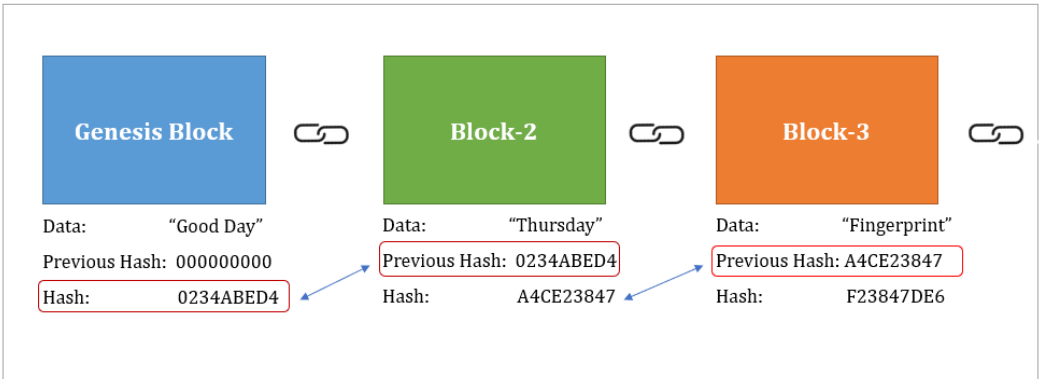


Figure 1:Blockchain Structure

As seen in **Error! Reference source not found.** Each block has its own unique hash [4], and it is calculated using all the transactions and hash from the previous blocks, thus forming the "blockchain", the latest block in the chain represents the total blockchain as one value, making comparing block validity easy.

As every block is dependent on the value of the last block, If a block in the chain is attempted to be modified, then changing any transaction in the chain will result in a completely new chain as it would

make the rest of the blocks invalid due to the changing of the hash value. Assume that block 2 is modified then it will be given a new hash value which would result in a change to block 3 due to requiring the previous hash of block 2 which would then break the blockchain.

2.1.2 Blockchain Design Trilemma

The blockchain trilemma was coined by Vitalik Buterin creator of Ethereum who proposed the three main problems with designing a blockchain, at this current time there is always one property that is forced to be sacrificed for the sake of the other two [5]. Retaining all aspects of these three features is extremely difficult without making any sacrifice to one of the properties: security, decentralization, and scalability. The current problem today is achieving all three simultaneously without any trade off.

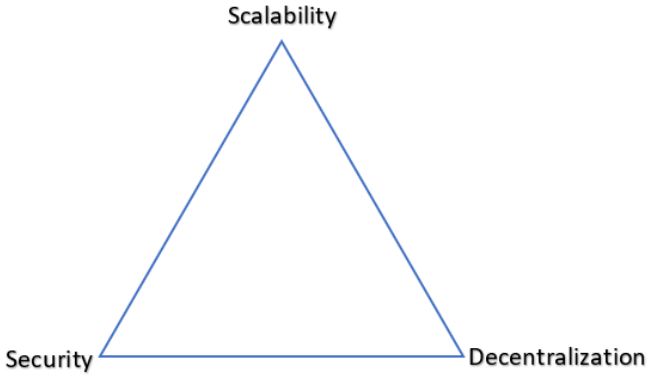


Figure 2:Blockchain Trillema

We can see this with the current situation with Bitcoin and Ethereum, although they are highly decentralized and secure, their scalability is sacrificed.

- Scalability: a blockchain which can process more transactions than a single regular node can verify.
- Decentralization: A chain which can run without any trusted entity, thus no centralized actors controlling the blockchain.
- Security: The blockchain can resit large percentages of nodes attempting attacks on the network typically anything about 25% of the nodes, ideally 50% anything below is not fine according to how Vitalik states a blockchain designs [5].

It's not necessary that this rule will make a blockchain never achieve optimum levels of all three properties however at this current time there is this problem today. Although there is current theories and new implementation developed however due to the infancy the project didn't cover this scope, two most popular methods to this date are sharding and layer 2 solutions which are planned to be implemented in the Ethereum blockchain, more on this can be read [6].

2.2 Consensus Mechanisms

Consensus mechanisms are a crucial role in blockchain design architecture maintaining safety and efficiency, as it enables the parties in a distributed network to come to an agreement between the data presented on the ledger. Consensus algorithms are designed to assume that some processes or systems will be down or unavailable and that some communications will be lost, as a result fault tolerant is a requirement.

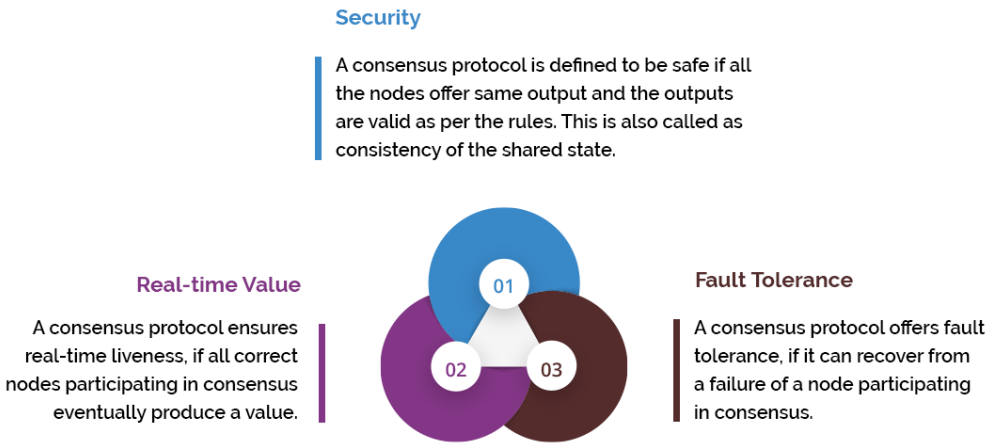


Figure 3:Consensus algorithm rules

As depicted in the figure above three rules of a consensus mechanism consist of security, real time value and fault tolerance. Without a central intermediary the network of participating users developing this system need to agree on the validity of what's being added to the ledger using a set of predefined rules A majority of the nodes on the network must reach an agreement, the effectiveness of implementing consensus mechanisms it is still a work in progress today [7].

Consensus algorithms also need to address the Byzantine Generals Problem which is that there will be some malicious nodes deliberately undermining the consensus process [8]. There are many types of consensus mechanisms, below is a few which will be examined throughout this project which aim to address the Byzantine Generals Problem.

2.2.1 Proof of Work

PoW(Proof of work) was first introduced in Bitcoin which involves scan for a cryptographic value that will and participants offering computing power to solve a hashed value such as with SHA-256(Used in bitcoin), the hash begins with a number of zero bits the average work required is exponential in the number of zero bits required and can be verified by executing a single hash [9]. To create a block hash small conditions, need to be met depending on the algorithm used, typically its finding a hash with a leading number of zeros.

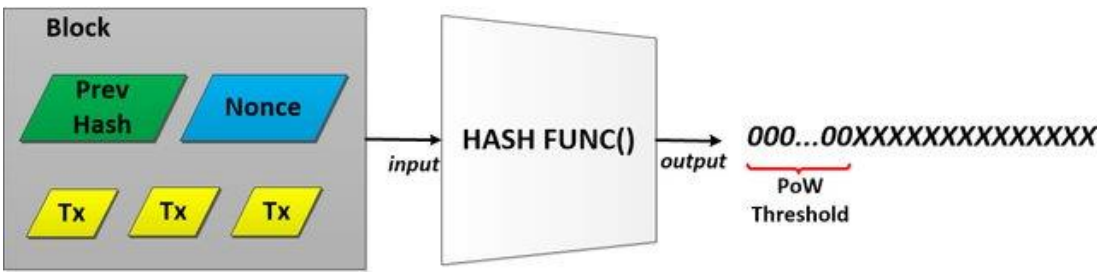


Figure 4:PoW Hash Steps

The main idea is having nodes spend computational power to solve expensive hash values before they can suggest a new block. The node who solves the problem first, mines the new block and broadcasts that message to the nodes in the network who then can verify the correctness based on the values in

the block. As we see depicted in Figure 4 [10], all new blocks are dependent on the previous hash values in the blockchain. The typical steps to require a new block are :

1. Miner will create the block header which includes timestamp(Tx),Nonce value, Previous hash
2. Miner adds a nonce(32-bit value) to the data and feeds that to a hashing algorithm – The nonce is the only part a miner can freely change and use to solve the hash
3. Get hash value
4. Test that targets hash
5. If met, then miner has generated a successful nonce that will add the block to the blockchain
6. If not met, then miner tries a different nonce and repeats back to step 2

The first node to successfully generate the correct hash will be rewarded based on the consensus mechanism, this process of solving the block hash takes huge amounts of computation as the network increases, each node competes to be the first to solve the block.

To protect against malicious attackers on the network for example trying to modify a single block in the blockchain this would change the block hash and make the whole subsequent blockchain invalid, it's feasible for an attacker to overthrow one block in a chain but as the blockchain increases the workload to overthrowing the long chain requires huge amount of computing power [11].

2.2.1 Proof of Stake

In PoStake (Proof of stake) is described as an energy efficient protocol, rather than ever node competing while using computational power to solve block hash, PoStake selects validators at random with a higher chance based on the amount of stake held in the network, validators will verify and confirm blocks in the network. The validators stake is used to incentivise good behaviour for example slashing the user's portion of their stake for things such as going offline or their entire stake for deliberate collusion [12]. The consensus mechanism removes the need to mine blocks and rather chooses to create blocks when selected to validate blocks, validators also get rewarded for attesting new blocks , attesting a malicious block will cause a loss in stake.

Threats of malicious attacks still exist within proof of stake, one of the most popular being the 51% attack but staking makes it riskier for attackers, as controlling 51% of the total supply is disincentivised by the costs of having to own the network which would also result in the value of the network decreasing overall. Whereas with a PoW 51% attack, it's more likely due to using computing power rather than total supply stake. They are stronger incentives to keep the network secure and healthy.

2.2.2 Proof of SpaceTime

PoST (Proof of space and time) proposed in the cryptocurrency Chia in 2018, their aim is to provide a greener alternative to PoW. PoST is very similar to PoW, instead of using computational power or staking, Proof of space uses storage space. Nodes in the network show that they have stored the data over a certain amount of time. Random nodes are selected to have their data read for verification, the more storage space you have the greater the chance of being randomly selected.

Chia a blockchain which uses PoST is a much closer alternative to the One-CPU-One-Vote like stated in bitcoins whitepaper [13], as the PoST resource as a trade-off between CPU work and space time, as the difficulty increases rather than scaling the computational power required like PoW does, PoST can increase the time period over which the data can be stored rather than increasing computational power [14].

2.2.3 Practical Byzantine Fault Tolerance

In PBFT (Practical Byzantine Fault Tolerance) used mostly in private blockchain networks, Each node in the network maintains an internal state regarding ongoing specific information and status of the network and when the node receives a message, they use the message in conjunction with the internal state to run a computation, this in return helps the node arrive at a decision regarding the validity of the message, after reaching its individual decision about the message, it's shared with all other nodes in the network.

The consensus decision in determined based on the total decisions submitted, if a node is missing in the network they can be given a default value as it can be assumed that the message for that particular node in the network is faulty, if the message was not received within a certain time limit, the algorithm allows the creator to assign a default response if the majority of nodes in the network arrive at the same decision [15].

PBFT has two potential problems relating to Byzantine Generals Problem, firstly all involved parties must reach agreement on the exact list of trusted participants and secondly the membership of such an agreement is set by a central authority. This addresses the Byzantine Generals Problem however the trade of is the decentralization aspect, these factors may not make such a consensus algorithm suitable for public blockchains, it can be extremely useful in private blockchains such as Hyperledger [16].

2.3 Smart Contracts

In 2015 the Ethereum blockchain was released and it first introduced smart contracts to a blockchain platform [17]. A smart contract is a digital contract that is stored and executed on the blockchain just like a regular transaction, however they are executed when predetermined conditions are met, typically used to automate the execution of an agreement so that all participants can be immediately certain on the outcome without any intermediary to oversee the contract.

Execution of the contracts typically following "if/when … else" statements that are written in code on a blockchain, these actions could include releasing funds to appropriate parties, registering a vote, sending and so on when conditions are satisfied. Once the execution has completed then the blockchain is updated, once the complete just like a transaction the contract cannot be changed, modified [18].

Smart contracts lead to the creation of many decentralized financial applications (DeFi) which introduced popular services like decentralized exchanges, loaning, and other systems like voting and non-fungible tokens (NFTs).

2.3.2 Domain Specific and General-Purpose Programming Languages

After the implementation of smart contracts on the Ethereum blockchain there has been many different programming languages used for smart contracts. The two types of languages a primarily between general purpose languages and domain specific languages, where the aim of using general purpose languages is to keep a familiarity to developers and use popular programming languages such as Go, C++, Java whereas domain specific languages are specially designed languages specific for the purpose of smart contracts, these languages aim to make writing smart contracts easier by adding tooling to the language to make development easier, one of the most popular languages in the smart contract development space is Solidity which is the main language used on Ethereum.

2.4 Ethereum Smart Contract Architecture

Ethereum is a well-established decentralized software platform with its own cryptocurrency called Ether, the platform enables developers to build decentralized applications on top of the blockchain using a DSL language called Solidity which is Turing complete. and permissionless blockchain making it accessible to anyone, all transactions, data and smart contracts for applications using Ethereum can be viewed publicly.

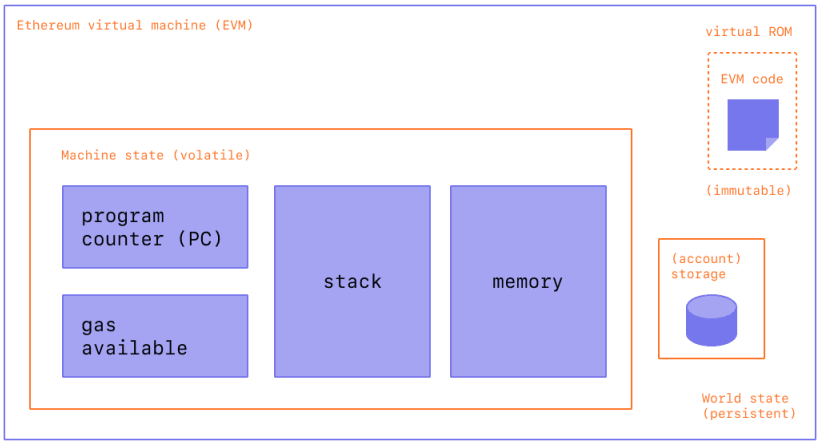


Figure 5: Ethereum EVM Architecture

Every node on the network is ran under the Ethereum Virtual Machine (EVM) to execute smart contract code and transactions. If the user wants to make a change to the state in the blockchain then they can do that by publicly announcing the transaction to the other nodes, the transaction will then be verified by the rest of the network using consensus mechanism.

Recently on August 5th 2021, Ethereum held an upgrade to the way their transactions work to make fees more predictable. Every block has a base fee, a minimum price per unit of gas for inclusion in this block is calculated by the network based on demand for block space. The base fee of the transaction fee is burnt, calculating this fee is as follows, where the tip is a priority fee paid to miners.:

Gas units (limit) * (Base Fee + tip)

To deal with malicious participants who aim to attack the blockchain or for hostile computational wastage in code like infinite loops, each transaction is required to set a limit to how many computational steps of code execution it can use. In Ethereum this is called Gas which refers to a unit that measures the amount of computational effort is required to execute specific operations on the Ethereum network. If the fee exceeds the balance of the sender, then the transaction is reverted however the gas is not returned it is sent to the miner [17].

2.5 Hyperledger Fabric Chaincode Architecture

Hyperledger Fabric is a private permissionless blockchain built for enterprise-grade platforms. Hyperledger Fabric requires participants to be enrolled through a trusted membership service provider(MSP) provided by the owners of the blockchain. They offer modular architecture allowing organizations to change and customize the consensus mechanisms and MSPs set in their blockchain [19].

Hyperledger Fabric main components:

- Organizations – Corresponds to real companies, are groups/members on the blockchain , they can create channels
- Channels – private subnet of communication between two or more specific network members, these execute transactions on the network, these hold smart contracts/ chain codes
- Peer – Individuals owned by an organization, they can join channels, have their own membership identity given by the MSP provider, they can host smart contracts and hold ledgers. Every channel (network) has its own data in a separate ledger stored on peers. Applications connect to peers to invoke and query contracts.
- Anchor peer – is a member which can discover other organizations on the network and communicate that with the rest of the peers in the organization.

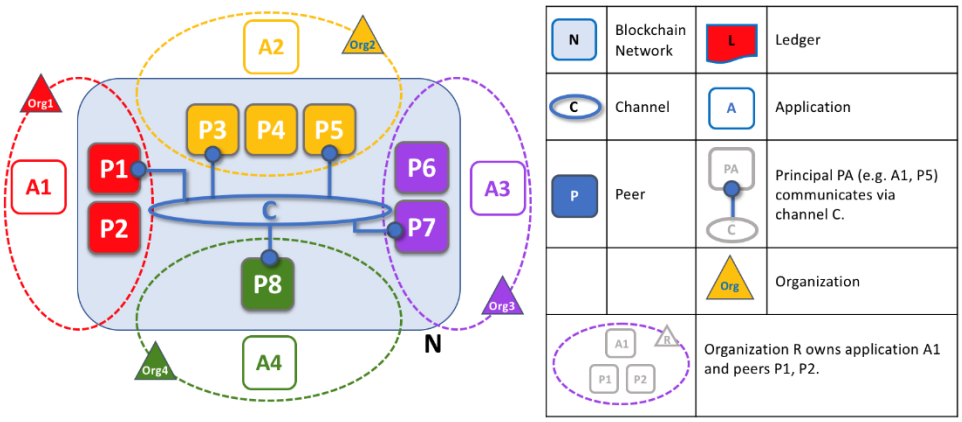


Figure 6:Hyperledger Fabric Hierarchy Diagram

In Figure 6, four organizations are dedicating eight peers to form a network. Channel C connects five of these peers in the network (P1,P3,P5,P7,P8). The other peers owned by these organizations have not been joined to this channel but typically linked to at least one other channel. Applications developed by a particular organization will connect to their own organization's peers as well connecting to those of different organization [20].

In Hyperledger Fabric a smart contract is contained within chaincode, allowing multiple smart contracts to be inside one chaincode. When a chaincode is deployed to the organisations network the smart contracts are all accessible. A smart contract can be thought as governing transactions, whereas chaincode governs how smart contracts are packaged for deployment. Chaincode is written in general purpose programming languages like Go, node.js, Java.

2.6 Daml Architecture

Daml is a high level functional DSL language framework used for creating smart contract based distributed applications, enabling safe high level real time business logic helping developers focus more on programming the business process by cutting down time spend on dealing with encryption and blockchain, with GPL languages the complexity is much greater, and the amount of code required is at least 5 to 10 times more to achieve the same impact as a fit for purpose language like Daml. Daml provides developers primitives to safely describe concerns like data types, privacy and authorization rules without concerns about the final deployment target.

Daml uses a virtual shared ledger, governed by smart contracts, this takes the place of a concrete blockchain adding this layer of abstraction allowing Daml to accomplish a whole range of capabilities and advantages that are lacking in individual blockchain and distributed ledger technologies. They state that this allows them to decouple applications from the underlying storage and consensus, making Daml contracts portable and enabling privacy and interoperability without the trading off consistency.

Figure 7:Daml Virtual Shared Ledger

The virtual ledger expresses who may write what events, event sourcing means the current state of the system can be computed from the log of past events building privacy due to not requiring a global event log. Stakeholders on Daml have a consistent view on a virtual shared event log. Actions taken can be seen on a need to know basis under the hood, creating a system which participants have a consistent view of the global system state consisting of the data they are entitled too [21].

3.0 Design Process

This section includes the design process and planning on the applied part of the project, including testing consensus algorithms and documenting my findings along with designing smart contracts on multiple blockchains. The main documentation can be found throughout this section while a full detailed step by step process can be found in the Appendix.

3.1 Blockchain Consensus Mechanisms

The consensus mechanisms didn't require any design, the process was to download and install the specific blockchain ledger relating to the consensus of choice and then attempt to participate as a node on the network. Then documenting the process, setup, and findings. A brief introduction into the process will be outlined below.

3.1.1 Proof of Work

To implement the proof of work on the bitcoin blockchain, the process couldn't be completed on a consumer computer due to the size of the network and large energy requirements. When installing the bitcoin ledger, I wasn't able to locate a valid mining option as the bitcoin core (provided by the bitcoin organisation [22]) as this method involved running the mining on the CPU which worked before bitcoin became widely popular, computational requirements are much higher and requiring the use of GPUs.

The work around, was that I joined a popular mining pool called NiceHash [23], which is another way for people to connect and combine their computational power and then the mining pool will split the rewards to all participants in the pool.

3.1.2 Proof of stake

To become a proof of stake validator on the Ethereum network there was a requirement to have a currency of 32 Ether, due to such high-cost requirements to opt as a validator it was best to test this process on their Goerli test network which simulates the Ethereum proof of stake chain.

3.1.3 Proof of SpaceTime

The plan was to use the Chia blockchain to operate the proof of spacetime consensus. There weren't many requirements for this apart from having at least 239GiB of free hard drive space.

3.2 Smart Contract Design

Two smart contacts where designed and tested against three programming languages, Ethereum Solidity for public permissionless use cases and Hyperledger Fabric Go which is a private permissioned blockchain built for enterprises as well as Daml a multi-platform smart contract language.

3.2.1 Asset Transfer Contract

To design the asset transfer contracts there was some changes made between each language. For example, for the Hyperledger blockchain the asset transfer didn't require designing an appraiser and inspector as planned due to Hyperledger being private and permissioned, it was built in with the blockchain itself which would allow the organisation running the contract to manually do this off chain.

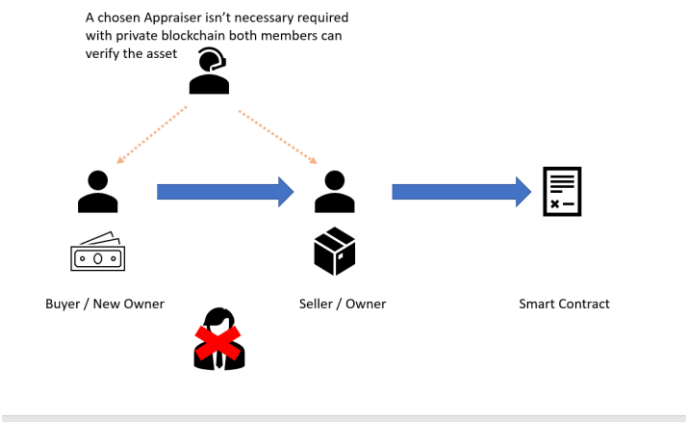


Figure 8:Hypeledger and Daml Asset Transfer Design

Whereas with Ethereum since its public permissionless and anyone can join it felt like a requirement for security aspects to implement an inspector and approver.

4.0 Implementation

This section documents the results from the designs, including screenshots of consensus algorithms, smart contract tests. The comparisons and discussions on each will be discussed in the next section.

4.1 Consensus Mechanism

4.1.1 NiceHash Mining Pool Bitcoin

Joining the PoW mining pool was surprisingly easy taking less than a minute to join following the NiceHash instructions.

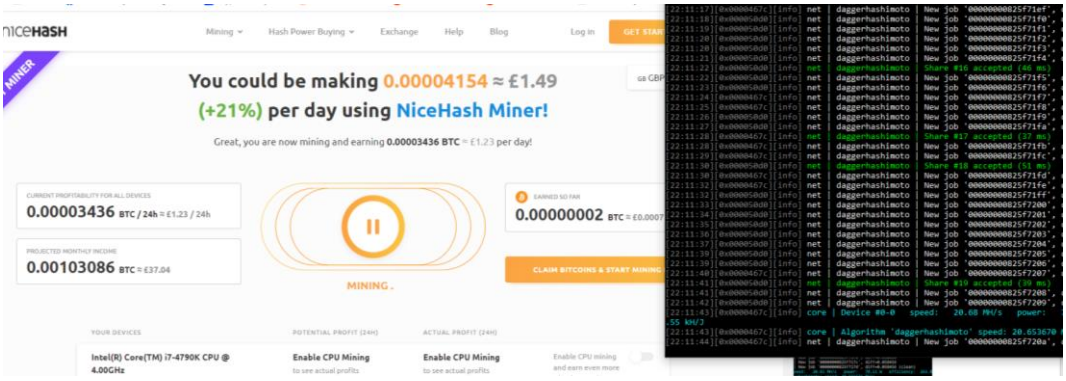


Figure 9:NiceHash mining pool operator

The mining pool helps solve the hardware limitations Bitcoin PoW unintentionally created, due to the high popularity, computational power requirements in the network are extremely high. Although this comparison is subjective due to using a mining pool rather than individually running the consensus algorithm, the software interface was user friendly, and the mining process was working in a few clicks. We can see this in the image above, the interface gave a terminal which displayed information on the mining process.

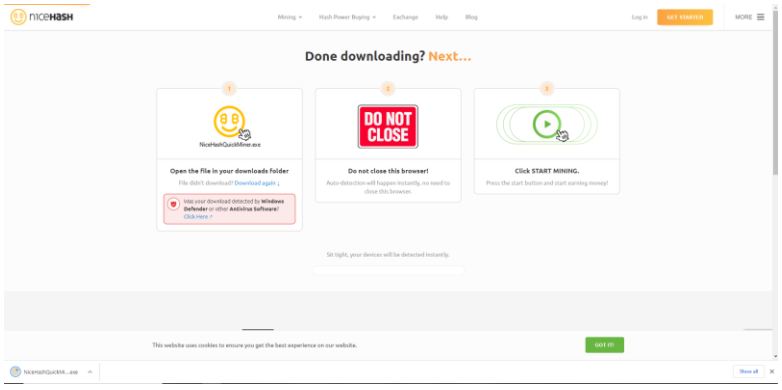


Figure 10:NiceHash Executable Steps

Note: This is a third-party software, there is possibility that this type of software from could be malicious be careful choosing a mining pool, an executable file was required with setup which will prompt open the mining terminal as shown in **Error! Reference source not found.**.

4.2.2 Ethereum 2.0 Proof of Stake Validator

Running proof of stake was surprisingly low maintenance after setup, the issue occurred was with the setup requirements:

- Having to run a beacon node to connect to Ethereum Poof of Work chain
- High number of steps required
- Basic knowledge of the terminal required

As mentioned, the Ethereum test network Goerli was used to participate as a validator in the network, there is a 32 Ether requirement to join and stake, the test network follows the exact same process. The whole process is documented in the Appendix, showing all commands used.

Step 1 : Download Eth1 endpoint for Eth 2 client

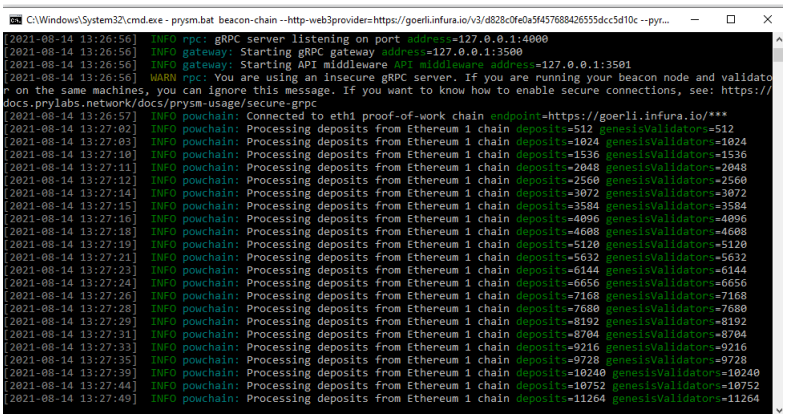


Figure 11:Beacon install connecting to infura server (Eth 1 endpoint)

Infura [24],a third party popular free tool for blockchain developers was used to create an Ethereum 1.0 endpoint which is required for the beacon node, it was easier to setup rather than manually installing an Ethereum client like Geth, networking and port forwarding setup was required whereas infura didn't.

- The beacon node brings proof of stake to the Ethereum blockchain by connecting to the Eth1 endpoint.

Step 2 : Choosing Eth 2 Client, Send Ethereum to the Eth 2.0 client

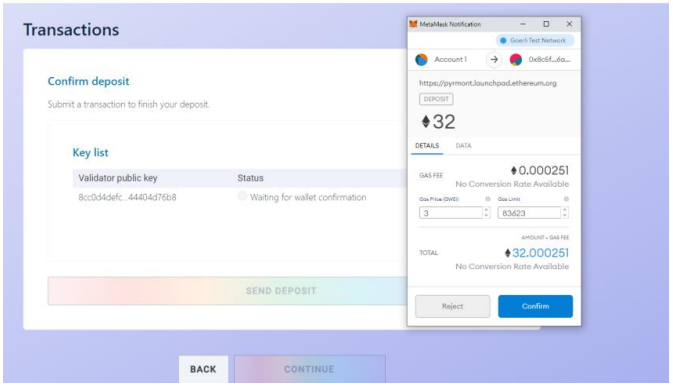


Figure 12:Transferring Eth to ETH2.0 Staking

To connect to the PoStake chain, holdings from an Ethereum wallet required a deposited to an Ethereum 2 address created from the setup of the Ethereum 2.0 client, in this case was Prysm, installation instructions can be found on their website [25].

Step 3 : Run Validator node and await commands

Unfortunately a known problem with the Goerli testnet and Prysm client was identified when trying to run the validator, once the beacon was installed, there was a problem connecting the validator to the beacon node: “You can either run your own updated geth node with the fix or wait till it’s an official release and infura updates its nodes with the fix” this issue here [26].

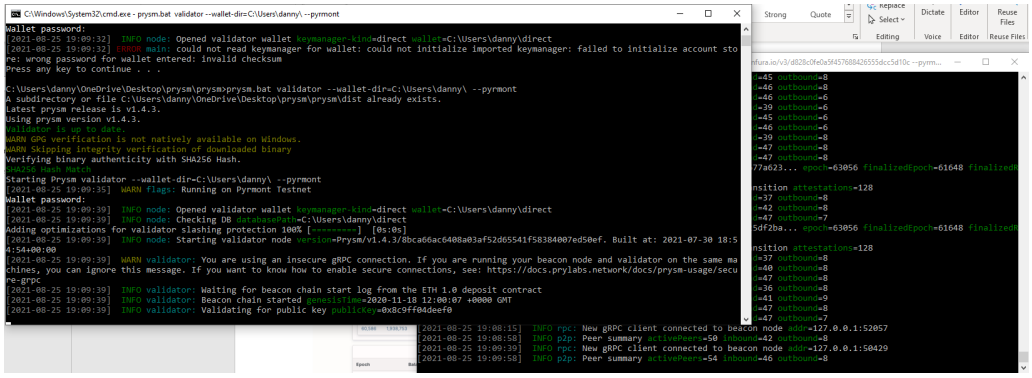


Figure 13: Terminal running Validator and Beacon chain

Figure 14: Validator Active Slashing

Since my validator address activated on the test network and I am not running the validator, there is a penalty for being offline. You can view this here on the Pyrmont beacon tracker [27].

Figure 15:Optimal Validator Client from Prysm Documentation

Taken from the Prysm documents [28], this is what should have been shown when the validator was working as intended and fully connected to the beacon chain. Unfortunately, this wasn’t the case due to complications out of my hands.

4.1.3 Chia Proof of SpaceTime Validator

Proof of SpaceTime was very user friendly and its interface was developed extremely well for non-technical users. The documentation was easy to follow as well as the process. The steps are as follows:

- Install Ledger
- Plotting (SSD)
- Farming (Hard Drive)

We first plot the data which requires a lot of time it's advised to use an SSD for this due to increased read and write speeds, plotting and updating the ledger can simultaneously, also plotting can be done offline, if plotting is stopped it must be restarted as it needs to be done in one process without interruptions.

- Location is important when plotting because you want high read/write speeds, slower hard drives will take longer to plot.
- At the end of the plotting process the contents of the temp files are compressed (~100Gb) and sent to the location you intend to farm from. Typically, a large cheaper hard drives performance does not matter for farming.
- SSD plot with temporary files will be cleared and can be reused again for another plot.

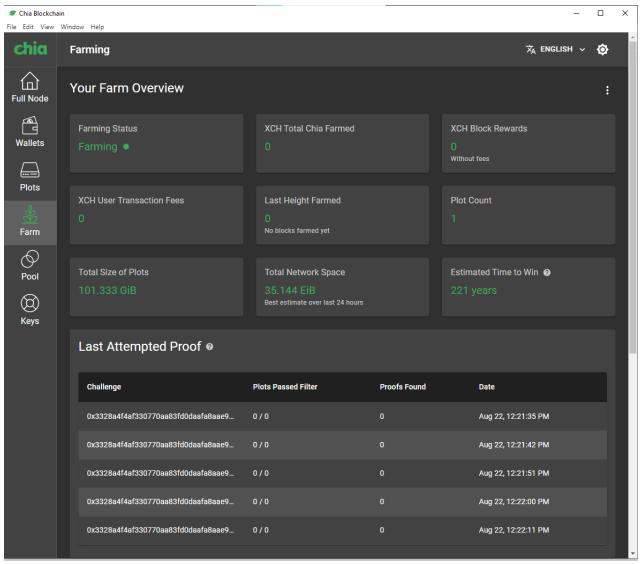


Figure 16:Proof of SpaceTime Interface

As shown above the estimated time to earn rewards as a node operator running one plot of ~100GiB currently takes around more than 221 years which felt unreasonable and underwhelming. In order for the nodes to be worth it , you would need a large amount of free hard drive space in order to farm.

The software design Chia brings for their consensus with a friendly interface was by far the easiest to operate for beginners while also having the option to use the command terminal if required.

4.2 Smart Contract – Asset Transfer

The primary smart contract discussed in this report was asset transfer contract, simulating how someone such as an enterprise would use a smart contract to transfer ownership of assets. There was also a ERC20 Token design this will be provided in the appendix.

4.2.1 Hyperledger Fabric Go Language

Platform

Hyperledger fabric has a few requirements:

- Git,cURL,Docker and Docker Compose installation
- Go installation and configured
- Installation of the Hyperledger Fabric docker image and dependencies.
- Download the Hyperledger Fabric samples which has a test-network and pre-built examples to deploy and test chaincode.

Hyperledger has the necessary documentation [29] , the process is documented in the appendix, once complete creating chaincode contracts was possible as the Hyperledger Fabric packages where successfully installed.

```
package main
import (
    "bytes" "crypto/sha256""encoding/json"
    "fmt"
    "github.com/hyperledger/fabric-chaincode-go/shim"
    "github.com/hyperledger/fabric-contract-api-go/contractapi"
//Provides the smart contract api interface
)
const (
    sellerPrice = "S"
    bidderPrice = "B"
)
//Init SmartContract
type SmartContract struct {
    contractapi.Contract
}
```

Figure 17: Go Hyperledger Imports

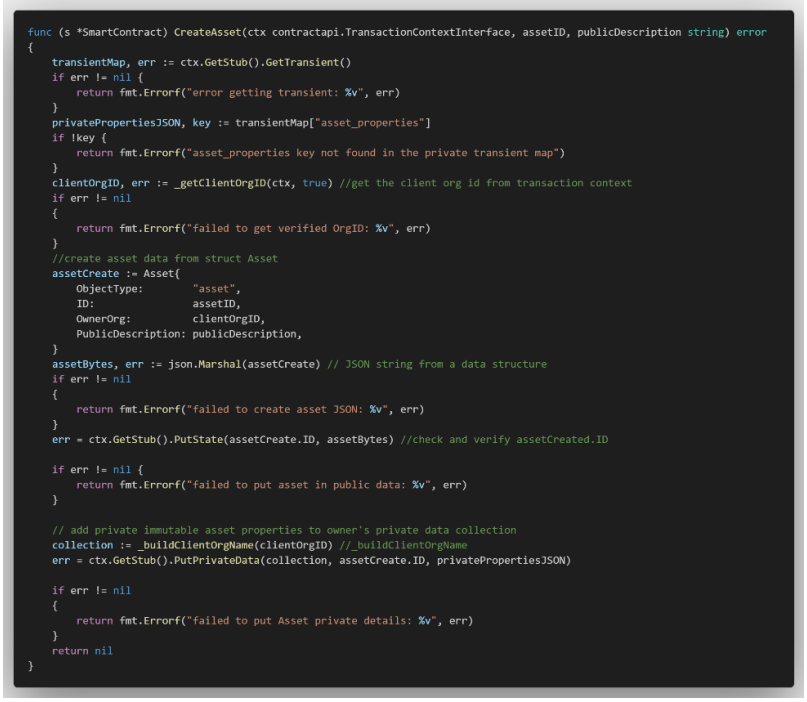
Development

Two important packages:

- fabric-contract-api provides contract interface, a high-level API for application developers to implement smart contracts.
- fabric-shim provides chaincode interface, a lower-level API for implementing smart contracts. Used to access state variables, transaction context and call other chaincode [30]

ctx contractapi.TransactionContextInterface: performs two tasks, first it allows developers to maintain user variables within smart contracts. Secondly, it provides Fabric API tools allowing operations such as detailed transaction processing, querying, or updating the ledger.

- ctx.GetSub() : accesses the ledger and requests to update the state to ledger
- ctx.PutState() : modifying data by changing state, logged to the ledger
- ctx.GetTransient() : Transient data is private to the application-smart contract interaction, private data



```go
func (s *SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, assetID, publicDescription string) error
{
    transientMap, err := ctx.GetStub().GetTransient()
    if err != nil {
        return fmt.Errorf("error getting transient: %v", err)
    }
    privatePropertiesJSON, key := transientMap["asset_properties"]
    if !key {
        return fmt.Errorf("asset_properties key not found in the private transient map")
    }
    clientOrgID, err := _getClientOrgID(ctx, true) //get the client org id from transaction context
    if err != nil
    {
        return fmt.Errorf("failed to get verified OrgID: %v", err)
    }
    //create asset data from struct Asset
    assetCreate := Asset{
        ObjectType:        "asset",
        ID:                assetID,
        OwnerOrg:          clientOrgID,
        PublicDescription: publicDescription,
    }
    assetBytes, err := json.Marshal(assetCreate) // JSON string from a data structure
    if err != nil
    {
        return fmt.Errorf("failed to create asset JSON: %v", err)
    }
    err = ctx.GetStub().PutState(assetCreate.ID, assetBytes) //check and verify assetCreated.ID

    if err != nil {
        return fmt.Errorf("failed to put asset in public data: %v", err)
    }

    // add private immutable asset properties to owner's private data collection
    collection := _buildClientOrgName(clientOrgID) //_buildClientOrgName
    err = ctx.GetStub().PutPrivateData(collection, assetCreate.ID, privatePropertiesJSON)

    if err != nil
    {
        return fmt.Errorf("failed to put Asset private details: %v", err)
    }
    return nil
}
```

Figure 18:Go CreateAsset Example Function

Testing

A lot of error checking involved to check for vulnerabilities, it's common when calling transaction context to set two variables one which includes an error for example:

```
transientMap, err := ctx.GetStub().GetTransient()
```

transientMap will hold the data, err will hold a value if there is an error found, for example is the err not equal nil then an error has occurred getting the Transient data, this err is helpful when debugging because you can return the error values, it also reduces the risk of possible attacks if there are errors within the contract code.

Two organization were implemented in this contract, Org1 which was the original holder of the asset and Org2 who is looking to purchase the asset. Private details could be created unlike Ethereum, which is necessary in some business situations limiting access to outside organizations, Hyperledger's API made it easy to limit access using function calls such as GetTransient(). Similarly, in this example functions such as:

- GetAssetPrivateProperties : this stores all the data to be viewed by organization 1 but organization 2 cannot
- ReadAsset : Reads the asset data public to all organizations listed on the contract

Deployment

Once the smart contract was implemented and deployed to the blockchain, we can query and invoke commands like how we call function on Ethereum. Unlike Solidity, commands to query and invoke functions where much longer, multiple conditions and flags must be set.

To Deploy the contract the command:

./network.sh deployCC -ccn secured -ccp ../asset-transfer-secured-agreement/chaincode-go/ -ccl go -ccep "OR('Org1MSP.peer','Org2MSP.peer')"

Query command example:

peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"GetAssetBidPrice","Args":["asset1"]}'

This example calls the function GetAssetBidPrice

To invoke a command its similar to the query:

This updates an asset

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"UpdateAsset","Args":["asset1","This asset is not for sale"]}'

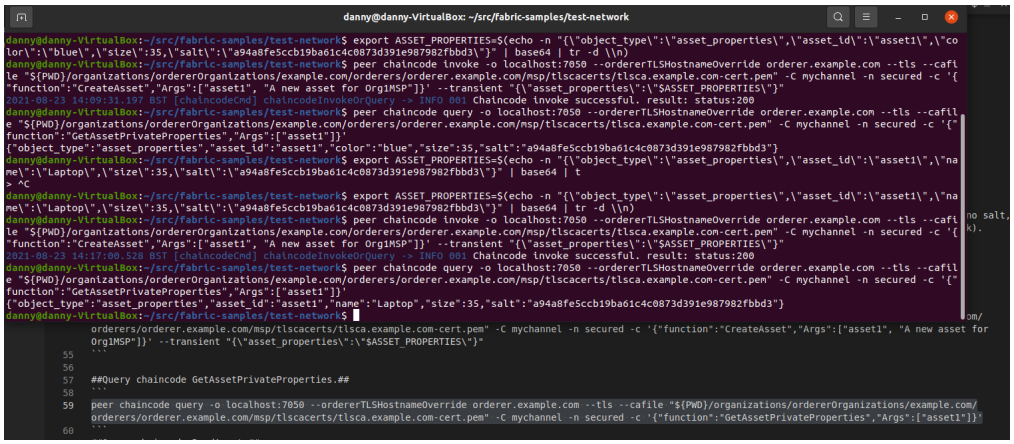


Figure 19:Invoke and query Chaincode Create Asset (Private details ORG1)

Figure 20:Read Public Asset

4.2.2 Ethereum Solidity – Asset Transfer

Platform:

Access to IDE this can be local installation or this case Ethereum has its own online IDE called Remix [31],which allows you to quickly get into developing smart contracts without needing to install dependencies. Remix can run locally on test networks and the main network, which makes it great for testing contracts.

Development

Developing using Solidity felt like JavaScript's syntax, with notable added features for smart contract design in Solidity, these include:

Custom variable types

- pragma: Solidity contracts require a version, for this case version 0.7 was used, the reason for this is to stop problems occurring when the Solidity language gets updates overtime.
- contract: collection of code defining the whole contract, holding functions and states that reside to a specific address on Ethereum, felt similar to object oriented programming.
- address: variable is a 20byte value representing Ethereum address which can, view balance, make address payable and transfer funds to that address.
- enums: helps reduce the possibility of bugs by restricting the variable to only hold a few predefined values, ensuring only these states can be accessed.

```
pragma solidity 0.7.0;

contract assetTransfer{
    enum StateType {
        Active, OfferPlaced, PendingInspection, Inspected, Apprasied,
        inspectorApproved, BuyerAccepted, SellerAccepted, Accepted,
        Terminated
        }

    StateType public State;

    address public InitOwner;
    string public Description; //bytes32
    uint public InitPrice;

    address public Buyer;
    uint public OfferPrice;

    address public Inspector;
    address public Apprasor;
    //Init conditions when creating contract
    constructor(uint256 price, string memory detail){
        InitOwner=msg.sender;
        InitPrice=price;
        Description=detail;
        State=StateType.Active;
    }
```

Figure 21: Create Asset Solidity

Test

Error handling:

- Assert(condition) which like JavaScript, if a condition is not met, call this function, and present an opcode and revert any changes done to the state. Used for internal error checking.
- Require(condition) if a condition is not met then it reverts to the original state, used for errors in inputs and external components.
- Revert()condition which aborts execution and reverts to the original state.

Deployment

Ethereum wallet such as MetaMask linked to Remix enabling contracts to be deployed to the Ethereum blockchain, in this case the Rinkeby test network was used for the purpose of testing.

When deploying the contract is costs gas to deploy to the network, there are factors relating to the cost of the deployment:

- Gas base fee
- Amount of byte code in the compiled contract, more means more storage in the blockchain
- Constructor in the contract (code run before creation on contract)
- Gas price fluctuation and limit if the network is busy the price will increase

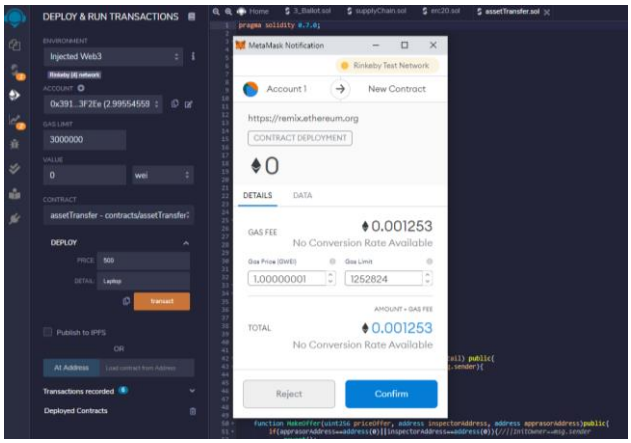


Figure 22:Asset Transfer Cost on Rinkeby Testnet

Once the transaction has been sent, it will take some time to be deployed to the network due to block times and consensus, also when making changes and updates to states in the contract e.g. updating asset, there was a significant wait time for a change to be updated to the blockchain. A gas price is paid, based on the number of computational steps required. Reading function is instant and requires no charge since there is no change to the state.

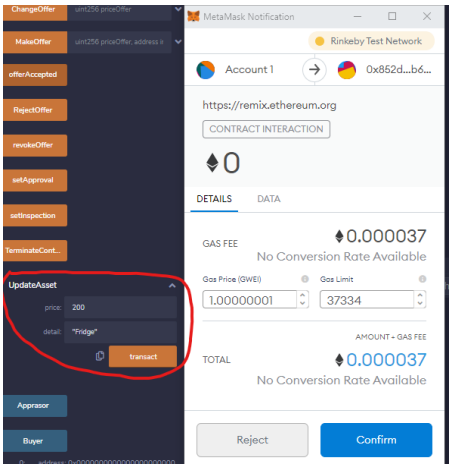


Figure 23:When updating a contract it requires a cost

4.2.3 Daml – Asset Transfer

Platform:

The requirements for Daml are as follows taken from their documentation [32], Daml called their extension Daml Studio:

- Visual Studio Code (only supported IDE)
- Daml Visual Studio Extension
- JDK 8 or greater
- Daml SDK installed following documentation

Daml Studio provided, tools such as syntax highlighting, real time feedback, information on hover to the IDE, which was useful when developing and testing code, it also helped the deployment and scripting process by creating an virtual ledger to audit the contract.

Develop:

Development was easier than expects mainly due to the well documented guide and the simplicity of the coding language.The structure was easy to follow and personaly found it much more enjoyable to create and read.

```
template Asset
  with
    owner  : Party
    approver : Party
    name   : Text
    price  : Decimal
    description : Text
  where
    ensure name /= ""
    signatory approver

    controller owner can
      nonconsuming GetPrice: Decimal
        do return price
      Transfer : (MakeProposalID, CheckAssetsID)
        with
          newOwner : Party
          ownerCheckAssets:  CheckAssetsID
          newOwnerAssets:  CheckAssetsID
        do
          updatedSellerInventory <- exercise ownerCheckAssets RemoveAsset with asset = self
          sale <- create MakeProposal with owner, newOwner, newOwnerAssets, ..
          return (sale, updatedSellerInventory)
```

Figure 24:Daml Contract Example

Notable types:

- template : is used to create a contract, we can think of this like a function or similar to how Solidity uses its contract which nests the contract functionality inside it.
- with : follows the names of parameters and their type, just like variables
- where : sets authority
- ensure : if condition evalute to true after ensure then create contract
- signatory : who must consent to creation of the contract
- observers : Parties who arnt signatory but still want to view the contract
- controller: who exercises the choice

- nonconsuming : makes a choice does not get archived, only the controllers and signatories of the contract can view all consequences of the action, useful when exercising a choice more than once.
- ContractId a: is the return type, archives the current Contract, creates a new one. Where a is the What it returns is a reference to the new contract, in the form of a ContractId a is the tempate name
- <- : means run the action and bind the result
- exercise : takes the ContractId, value of c where c is a choice in the template. Executed on server.
- create : updates the ledger to server, createcmd builds up a list of commands to be send to the ledger, where create builds up a more flexible update executed directly by the ledger.

Test:

Running scripts to test the functionality felt difficult, understanding the documentation took me longer to grasp. Once scripting was understood it felt easier checking errors in the contract due to the testing locally through the built-in ledger as seen in the deployment, locally testing felt useful as once achieved can be sent to multiple supported platforms like Hyperledger Fabric [32].

Notable commands:

- Import Daml.Script : required API for running scripts to test scenarios.
- Import Daml.Assert : requirement to assert values when testing e.g name === “laptop”
- Script : run script test different parties submitting transaction to check if the templates are working as intended.
- Submit : is an action to the ledger(create or exercise)
- createcmd and exercisecmd : used to build on the client side of the ledger where without cmd the server are executed directly on the server. Useful for privacy, sharing specific data publically while keeping rest private.
- Commands (ContractId makeProposal, ContractId makeProposal) : proposals created similar to updating, individual commands do not depend on each other.

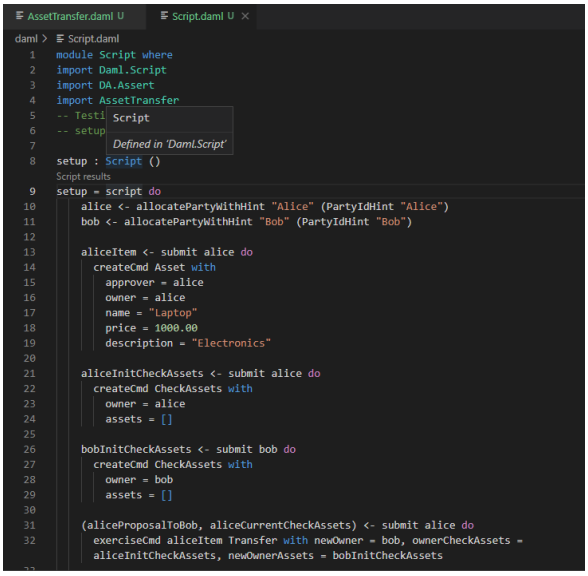


Figure 25:Daml Testing Script

Deployment:

Deployment to a blockchain network wasn't pursued in this project, testing locally felt enough to understand how Daml works. The local ledger displays transactions performed from the scripts. Archiving is when a new contract has been created and can be viewed by checking the box on the ledger [32].

- S :signatory of contract
- O :observer of contract
- W :witnessed creation of the contract, as they are the actor on the exercise creating it.
- D :witness to the exercise that resulted in a fetch of the contract

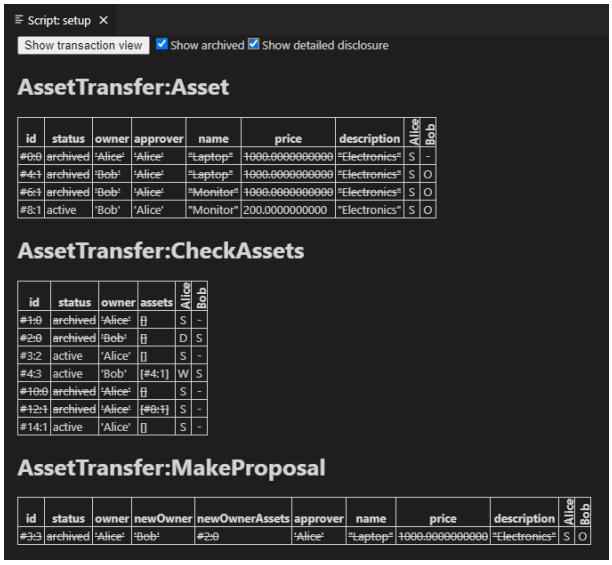


Figure 26:Daml show transaction ledger table format

5.0 Discussion

This section discusses findings achieved from the implementation section, comparisons will involve consensus algorithms, smart contracts and blockchain trade-offs between Hyperledger and Ethereum.

There will be comparisons made throughout the discussion with references to their advantages and disadvantages, to allow the reader to conclude their own view on which consensus and smart contract language they think is best, I will give my personal opinions and reasons why throughout the discussion with references to sources.

5.1 Consensus Findings

Findings based on two positions: blockchain operators and consideration for blockchain developers:

- Firstly, how easy is it for participants to join the network as a consensus operator for each specified blockchain tested, the metrics following this point is measuring the software user interface, hardware requirements for setup, documentation, and the steps required to execute the process.
- The second area looks at benefits and limitations of using this type of consensus protocol which a blockchain developer may consider: factors relating to environment impacts and the blockchain trilemma.

	Software UI	**Hardware Requirements**	**Documentation**	**Execution Process**	**Environmental factors**
Proof of Work Bitcoin-NiceHash	Good	High/Costly	Great	~60seconds	Very poor
Proof of Stake Eth2.0	Poor	Very Low	Good	~12 hours	Very low
Proof of SpaceTime Chia	Great	Low - Medium	Great	~12 hours	Good- poor

PoW Findings

Running bitcoin mining on a personal computer is impossible as competing for blocks is far too difficult with conventional computers, the probability of individually mining a block is so low that it would take years, which is infeasible when factoring costs of electricity and hardware lifespans [33]. One note to point out is that Bitcoin also isn't ASIC resistant (Application specific chips) resulting in specialized mining hardware creation for large corporations with access to premium hardware and electricity. Ethereum and many other blockchains who use PoW have combat this by implementing ASIC resistance algorithms into the consensus protocol to attempt to make mining more accessible to more individuals in attempts to increase decentralization, reduce mining companies and combats the handful of companies which produce ASIC hardware, removing potential control they can impose [34].

A feasible alternative is joining mining pools, while it promotes centralization by having the organization confirm the blocks using the combined computing power from users [33]. It was surprising how easy the mining pool was to setup, I thought it would have required extra steps due to it being a "Third party", it was far easier and was prompt with my estimate total reward per day. A big concern with mining pools is the possibility for attacks on the network which is theoretically possible for bitcoin as the increasing number of participants joining mining networks, it has proved possible in recent 51% attacks on other blockchains like Ethereum Classic [35] .

Although PoW is highly environmentally unfriendly, researching PoW in Bitcoin lead to interesting facts about how Bitcoin is mined, While the consensus is largely bad for the environment as reports

of bitcoin mining consumption being equivalent to small countries like Sweden (Consuming around 110Terawatt hours per year) [36]. Some nodes on the network use renewable resources to mine bitcoin, estimated amount equivalent to 30-35% of the network with a large majority using hydropower as the renewable source, estimated by research paper on bitcoin energy consumption [37] and a web report stating similar estimates at around 39% based on data from Cambridge centre for alternative finance [38].

PoStake Findings

Ethereum2.0 has a 32 Ether(~$100,000 Sept 2021) requirement per validator, after testing and research I found an important reasonings why there is a minimum limit to stake. This consideration was intended as a trade-off, risking higher centralization to lower the overhead in the network which could cause congestion from having too low of an entry point per validator. A low barrier to entry results in a difficult algorithm to design while also a lower reward distribution for operators. As mentioned on EthHub founded by Ethereum community members [39]. The higher barrier to entry makes the network more secure as per 1 validator is 32Eth it is much more expensive and difficult to attempt to attack the network even for staking pools, as its unlikely for one user to control a significant number of validators due to the cost being so high, it would cost significantly more that PoW and the risk with slashing for acting dishonest disincentives the attack from acting dishonest as discussed on the Ethereum Reddit [40].

A consideration to point out for operators using staking pools is that if a network outage or dishonest validator, all staking participants would receive penalties in slashing, so there is an increased reliance and trust on the staking pool to act as an honest validator [40]. However, with the high incentives for staking pools to act honest (as they take a percentage from stakes) it is unlikely for a staking pool to act intentionally dishonest.

An issue with high validator cost in PoStake leads to multiple sources mentioning the fact that the rich get richer [41], we see the 32Eth limit per validator equates to at $3400 per 1Eth, more than $100,000 worth of Ethereum. The issue then leads to an incentivization in staking pools as the majority of people wouldn't want to stake such a high amount to become a validator, thus users would have an incentive to join staking pools which would allow them to stake less and earn percentage based rewards, which maybe a concern when considering consortium companies like public exchanges holding large amounts of people stake, it opens room for targeted attacks and governments control discussed on the Ethereum forum [42]. Although the same issue occurs with PoW mining when factoring in hardware costs people are incentivized to use mining pools as they are easier and have a lower barrier to entry. Although staking pools can lead to increased centralization the official Ethereum website points out that staking is more decentralized due to the fact of economy of scale, allowing for increased participation with staking pools and more nodes doesn't mean increased percentage return unlike with PoW mining [12] and PoST farming [43] .

One weakness found from testing Eth2.0 PoStake was that it felt problematic and long to setup, extra steps such as the beacon node to bridge between the Ethereum PoW chain and the new PoStake chain, knowledge of the terminal was necessary, making it more challenging for non-technical operators. Although this problem was mainly due to the setup between the two chains as Ethereum transitions from their PoW chain to a PoStake chain.

The official Ethereum website points out similar discoveries to my findings in relation to PoStake, which that PoStake provides better energy consumption as to be a validator only a stable internet connection and hard drive space to store the ledger is required so devices like a raspberry pi can run

a validator, creates a lower barrier to entry. PoStake has a stronger support for sharded chains which results in a significant scalability increase as discussed here [12].

PoST Personal findings and comparisons

A newer consensus algorithm compared to PoW and PoStake with lesser research, the consensus aims at provider greener alternative to PoW and unlike PoStake, PoST aims to keep the use of scarce resources (storage space) making it much more available to everyone in hopes for higher decentralization as mentioned by Chia [44]. Instead of focusing only on computing power like PoW or raw storage space, an implementation of both is used balancing between both CPU cycles, storage, and some amount of time to verify.

Chia mentions that anyone can easily participate in the network by dedicating storage space, while this is true, to earn any realistic rewards a large number of storage space is required experienced with testing I found using 100Gib to farm, takes estimated reward of 200 years, an average user is most likely to have an estimated around 500Gib to 1TiB of storage [45], even with a couple hundred more gigabytes of storage the estimated reward is underwhelming. This is most likely due to how Chia scales work difficulty, by dynamically adjusting to achieve 32 blocks within a 10minute target time similar to PoW bitcoin, the farming difficulty is adjusted based on the amount of network space, if blocks are released too fast difficulty increases, as farming competition increases farmers should expect rewards from storage to go down ([43]).

Farming pools

The difficulty increase creates is an incentive for individual operators to join farming pools or invest heavily into hard drive equipment and forming investing into farming rigs like PoW mining setups although being much more environmentally friendly and less expensive. The incentives to join a farming pool to earn faster rewards, create a higher centralization aspect similar to the problems with PoW articles describe centralization issues from mining pools [46] [33]. The Consensus also has lower barrier to entry (storage space) which could possibly be worse than PoW mining pools which require a higher barrier (GPUs), its yet to be seen as PoST is still in infancy but when we see popular farming pools on Chia, iHpool on 20th September 2021, they currently control approximately 50% of the total network capacity [47] which is a very troubling security concern, people also raised this concern about pools on chia's forum [48]..

Chia does a great join in creating a highly accessible consensus, in theory the consensus promotes a higher decentralization in comparison to PoW and PoStake but when analysis various pools (mining,staking,farming), having to low of a barrier to entry such as storage devices can open the network up to a higher chance of attacks, its less risky than PoStake and easier than having to use computing power in PoW, participants can choose to join pools easier with more accessible hardware to earn faster as evidence from of this happening with iHpool [47]. When comparing PoStake both have a low entry to access but PoStake uses a 32 Eth limit per validator which makes it much harder to control most of the network however as mentioned with Ethereum PoStake design, they sacrifice some decentralization in hopes for a more secure and scalable consensus protocol, unlike PoST the lower barrier to entry creates higher difficulty and longer estimated reward times as operators aim to increase their network capabilities by adding more storage to increase their chances in being randomly selected to farm, it creates an inventive to join pools to receive rewards more often with lower investments.

E-Waste

Chia PoST consensus has concerns with hard drive lifespans, while testing the algorithm I came upon multiple sources discussing concerns about the lifespans of hard drives [49] [50] which seems to impose an weakness in the principle of environmental friendly Chia network, although Chia is in fact a greener alternative to PoW [44] , Chia does not generate anywhere near to the amount of wasted energy PoW has, there is some concerns raised with high amounts of e-waste created from burnt out storage drives as while generating plots puts strain on the SSD drive.

Chia have recently addressed the points while I have been investigating, on August 16th 2021 [51] Chia mentioned that plotting does require high amounts of write speeds (~1.3TiB of writes) per k-32 plot. While addressing this they mention that SSD endurance is important because SSDs will generally be at 100% duty cycles writing all day. Chia specifically state: "A mixed use or high endurance data centre or enterprise SSD is the best choice for plotting…. Consumer NVMe SSDs are generally not recommended due to the lower endurance ... They do not perform well under heavy workload … the lower rated endurance in TBW will result in faster wear out".

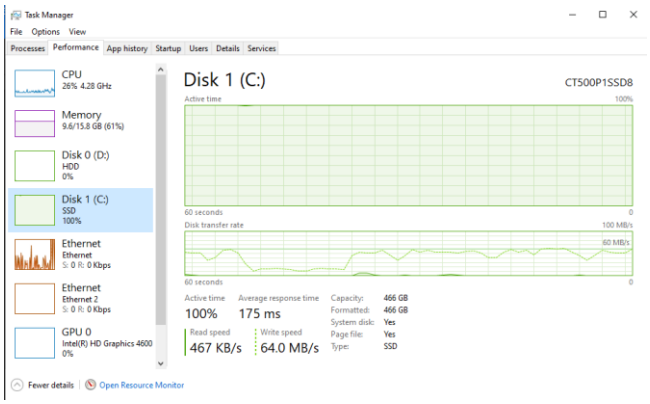


Figure 27:Testing Performance while plotting

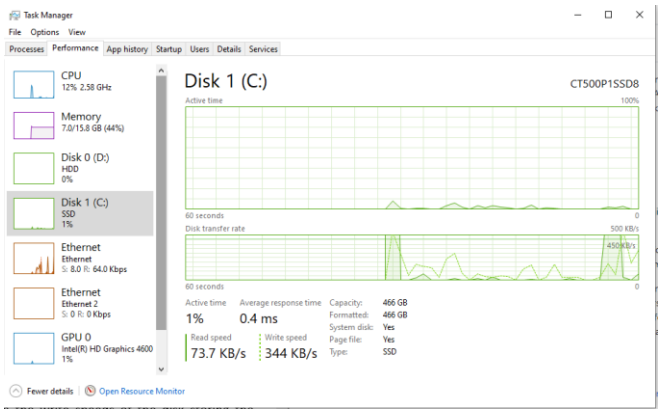


Figure 28:SSD Performance IDLE

Verifying the issue Chia addressed, the SSD active-duty cycles where peaking at 100% while plotting with write speeds of around 64.0MB/s on my Crucial NVMe, while not plotting we see that my SSD write speeds are sitting at the low KB/s while browsing the web. Concurring from this the evidence pointed that there is an area of concern relating to SSD lifespan and E-wastage [49] [50].

Chia's business whitepaper [43] mentions "We anticipate that chia farming will be more decentralized than PoW or PoStake and significantly less energy and resource intensive". A disagreement I found is that similar problems experienced with GPU prices in PoW are occurring with storage pricing highly

accessible to everyone the expected time for rewards are demotivating for operators who only have a few hundred GiB of spare storage space. I would argue that PoST is less decentralized consensus due to the incentivization to join farming pools [47] and the forums discussing larger pools [48, 52], however I see the argument that PoST aimed for higher decentralized due to highly accessibility , whereas PoStake on Ethereum requires a large investment to operate the network and PoW on bitcoin is impossible unless you are a large cooperation and investments in mining hardware.

Even though there some concerns with PoST, Farming is much more achievable to the average operator due to factors like ASIC resistance equipment and energy efficient farming. It is easier and cheaper to buy storage drives and build at home farming rigs unlike the other consensus protocols where this is impossibly without significant investment. Personally, PoST is an interesting concept and aims at tackling a big problem relating to environmental effects, centralization, and accessibility to everyone using scarce resources rather than currency tied to the blockchain itself.

Conclusion

Overall, from the findings detailed about I found PoStake to still provide the best environmentally friendly benefits in comparison to PoW and PoST, there is arguments for either consensus but overall the benefits kept leading me towards PoStake over the others. Maybe after there has been longer research on PoST to determine the potential e-wastage and security concerns it could be a good contended, especially when considering resources usage as you can argue that PoST can be more decentralized compared to PoStake due to available resources used. PoStake found to provide a safer consensus however it does sacrifice some decentralization as a trade-off for increased scalability and speed.

5.2 Smart Contract Languages

Findings based on smart contract languages primarily differences between both domain specific and general-purpose languages. Table below shows personal findings obtained from implementations:

	Solidity	**Daml**	**Go**
Language Type	DSL	DSL	GPL
Similarity	C++,Javascript,Object Oriented	Haskell,Python, Functional language	N/A
Documentation	Good	Great	Good
Syntax (features)	Great	Very Good	Okay
Readability	Good	Great	Ok
Resources Available	Great	Low	Ok (Hyperledger Fabric) Great (General resources)
Blockchain Platform	Strictly only on Ethereum virtual machine.	Multipurpose platform, focused on Enterprise solutions	Hyperledger, focused on Enterprise.
Private and public features	Only public , Solidity does support private features but wasn't designed for it, and the cost for private features is extremely high and difficult	Private with possibility to add public in the future	Primarily Private but public features can be done using Hyperledger Besu (running on Ethereum Client)

Figure 29 : Smart Contracts Comparison Results

Notable points:

- **Language Type:** Domain-specific Language (DSL) or General-Purpose Language(GPL)
- **Documentation**: Relating to the documentation provided by the blockchain supporting that language
- **Syntax** : The features found from testing contracts, functions which make creating contracts easier, faster to learn and similarity to other languages
- **Readability** : How readability is the smart contract based on length of code, syntax, and similarity.
- **Resources Available** : Aimed primarily at resources on smart contracts resources , measured by things like StackOverflow, Google, YouTube.
- **Score system** : Low – Good – Very good – Great, scores were referenced to give a brief overview of the category. While Great concludes that the category is clear, understandable, easy to follow.

5.2.1 Solidity Findings

Platform and Development

The setup requirements to create a Solidity contract was far easier with the addition of their online IDE Remix, which allows you to quickly create, test, debug and upload to the network. The accessibility felt rather high in agreement with sources from the official Ethereum Remix documentation [53] and on answers to the question why Remix on stackexchange [54]. Although Remix is great for quick development it was also mentioned that usually in final development scenarios frameworks like truffle are used over remix as they can execute JavaScript tests and integration for building smart contracts applications.

One disadvantage of using Solidity is that it's tied to the Ethereum ecosystem, while it is possible to tie the language for private blockchains, the blockchains much be complete with Ethereum (Hyperledger BESU for example) however the same issue occurs with the language being publicly viewable. Solidity also doesn't support privacy; anyone can access the contract and while there are private variables these variables which are tied to the smart contract meaning other smart contracts cannot access or modify the variables but their values can be read freely outside the smart contract [55].

Readability and Syntax

Syntax for Solidity felt highly similar to object-oriented programming and the syntax reminded me more of JavaScript and C++ similar to how Ethereum have spoken about developing the language syntax to be similar so developers can jump into creating contracts on Ethereum [55], Solidity was also designed around ECMAScript syntax to make it familiar for web developers [56]. One main difference is that Solidity is statically typed unlike JavaScript, meaning variables must be declared. More information on Syntax feature have been discussed in the Implementation section.

When we compare Solidity to the other two languages, it felt like Solidity has the most familiarity for developers due to the similarity to JavaScript syntax. Whereas Daml was highly custom and looked very unfamiliar to ordinary programming languages it had the better readability to Solidity to do its business-like contract logic. Hyperledger Go on the other hand looked more overwhelming compared to Solidity.

Solidity's readability and syntax felt like it fit in between Go and Daml where Go has low readability and Daml has condensed readable code structure. As discussed further in the comparison section a solid argument can be made that Solidity's similarity to JavaScript will work in a developers favour since JavaScript is one of the most popular programming languages, developers could have an easier time reading, communicating, and designing smart contracts for businesses.

Integration

Each contract on Ethereum has its own ABI (Application Binary Interface) which is the standard way of interacting with smart contracts in and out of the Ethereum ecosystem. Typically used to test contracts using JavaScript and design frontend applications by connecting to the smart contract ABI to act as the backend more can be read on the ABI from the Solidity Documentation [55].

- Web3.js: The library that helps in the connection between the Ethereum network via HTTP or the IPC network connection.

Security

Security in Smart contracts is a large topic of discussion, we will briefly discuss some security concerns with Solidity which is also mentioned in the documentation under Security Concerns [55]. Since Ethereum uses Gas for their solidity contracts it helps against malicious intentions like slowing down the network by running infinite loops, Gas fees and limits help the network by charging users a fee every time they make a change to the blockchain and a gas limit once this has been reached the contract will stop executing as discussed in the background section.

While the language has been designed to build securely smart contracts, there is always possible security concerns when it becomes to the developer designing the contracts, what make seem like a smart contract working as intend may not be the case internally. In Solidity it's much harder to solve due to the blockchain being publicly viewable.

- Everything in a Solidity contract is publicly visible even variables stated as private
- Loops that do not have a fixed number of iterations, e.g loops depending on storage values must be used carefully, due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the contract to be stalled.

Documentation and Resources

Following the documentation and various resources creating smart contracts on Ethereum was easier than expected from someone who is unfamiliar with the language due to the large community support around Ethereum and Solidity. Regarding the official documentation for Solidity [55], my personal experience felt that there was a clear and concise structure with many in depth examples which didn't feel like the case for Hyperledger. Although the documentation felt lengthy and long winded I personally though the documentation was helpful for referencing when I wasn't why and how to implement specific parts of my contract code, the various examples helped me quickly understand how to do specific tasks and provided explanations to variable choices.

I personally found it faster and easier to use other resources such as StackOverFlow and YouTube examples as there are many similar project examples and developers would give detailed explanations on design choices, whereas in comparison to Daml and Hyperledger where much harder to find and I had to rely more on the documentation they provided.

The Ethereum developer Reddit forum provided much support to questions from beginners and experts asking for advice, a beginner getting into Solidity asked what are the best ways to learn Solidity [57] and multiple users mentioned following the Solidity by Example in the documentation, and using other resources such as free courses and projects on YouTube to help learn, one user stated "the best learning I've done has been just trying to build various smart contracts and referencing both the official docs as well as existing high-profile projects when I'm completely stumped…", the thoughts seemed to be similar to how my experience was using the Documentation and resources. In comparison to Daml and Hyperledger it was much harder to find outside resources and help for specific problems, I had to rely more on the documentation they provided.

5.2.2 Daml

Platform and Development

As mentioned in the background Daml was run on its own local ledger allowing contracts to be tested before being deployed. The benefit in using Daml is the fact it is multi-platform allowing the smart contracts to be used on multiple blockchains which can become useful in enterprise businesses who use multi-platforms and ideally dealing B2B relationships.

One key difference between Daml and Solidity and Go is that Daml uses a UTXO model rather than account based like in Solidity, so workflows in Daml are a series of contracts rather than updating account properties in Solidity and Go [58].

Currently to use Daml Studio, it is only currently supported by Visual Studio IDE by installation of the extension, this is a small limiting downside to Daml however when using the extension Daml created for visual studio the development felt smooth, the extension provided necessary feel of more commonly developed language like Python, it helped correctly developing the contract by providing highlighting and detailing information about what the syntax is missing to make the contract work or current warnings from incorrect syntax. The extension made the development feel very smooth

compared to using Go and Solidity as it provided strict rules and guidance lowering the chances of creating any unintentional errors.

As discussed by a Daml Employee [59], in Daml every contract is by default visible to only a small set of stakeholders which makes it very useful for fixed consortium use cases similar to chaincode on Fabric and Solidity but also very suitable for use cases with dynamic stakeholders sets like public trading or settlement systems we see this with ASX stock exchange using Daml to handle public trading and settlements [60]. Similar to what a contract developer stated on the Ethereum reddit about his experience using Daml, he stated that Daml is excellent if the actors of the contract know each other, not so good if not, both languages have different use cases and excel at different areas [61]. As discussed further on.

Readability and Syntax

Personally, Daml looked had the best readability as the syntax look similar to English, the length of the code reduced significantly compared to the other two smart contracts, syntax was clear and easy to understand. Similarly to what was reported on the Medium website [62] discussing smart contract languages they mention Daml can usually make contracts in much less code "…~75% less Solidity Smart contracts or Go Chaincode code", similar to my findings although I argue that Solidity's code can also be a lot less in comparison to languages like Go from my implementations.

Daml excels at workflow structure model offering abstract thinking, according to research Daml stated [63] that they have tested the language on new Graduates with little experience in programming and given two days to create financial contracts on Daml, they felt the project was a success with all participants successfully creating the contracts, Chiara stated ""…once you know how to write a template in Daml you're pretty much done! It helps you focus less on the language and more on the finance." Another student (Varun) with programming experience stated, "Even if you're comfortable with another language, the little time it takes to learn Daml is a great investment ... the amount of coding that is abstracted away". A developer who has used Daml and Solidity stated similar saying Daml excels at process pipeline [61].

The experiments above were similar to my experience, Daml focused on the business logic rather than the coding aspect making it a great contender in businesses environments as its far easier to describe and develop contracts. Some suggested that Daml could be a good candidate in lawful smart contracts due to the readability [64].

Daml's high level domain language structure helped simplify the procedure of creating a contract, similar to a hierarchy, where the "template" is the name of the contract followed "with" parameter which lists the variable names such as owner, then "where" which adds the authentication, followed by "controller" which is similar to stating what the contract is going to do.

Security

Testing the virtual ledger helped a lot in the testing phase of the contract, while running scripts you could determine if things are working as intended as the ledger would auto update in real time.

The simplicity and structure of Daml made it easier to determine bugs and vulnerabilities as the language was very strict and required certain conditions to be met before the contract would correctly execute which reduces coding errors as discussed by Daml Employee [59]. Even though, I couldn't find detailed research on this aspect, we can effectively argue, improvements to security of the contract by having understandable and condensed code making it less open to development errors. On the

other hand, condensed code could make the development limiting and working around certain errors could become a problem.

Digital Assets wrote an article on Medium [65] with some valid points on the attacks with Ethereum Solidity contracts and when writing smart contracts today developers need to be aware of things such as re-entrancy attacks and over and underflow attacks and in financial industry lack of confidence in smart contracts is unacceptable and contracts shouldn't have to worry about things such as overflow errors. Although when dealing with public smart contract there will always be security concerns as the likely hood of attackers are higher than using a private blockchain.

Documentation and Resources

Daml had one of the best Documentation from my experience, the mix between in depth explanations and brief outlines in code made it much easier to learn and understand. Although when looking for extra resources online one downside I found is the lack of online resources and example projects. Although one main reasoning could be due to the Daml being used for enterprise solutions and typically some B2B's won't want their smart contracts being publicly viewable. Lack of resources could argue that the language is not well used by developers and pointed out by a contracted developer discussing the language on Reddit mentioned about working on Daml projects and there was lack of resources, while trying to find resources on Daml I felt like I kept running into the same examples just repeated, a developer who uses Daml said "..Ditto resourcing … We actually use Daml quite a lot, but we have zero chance in hiring for Daml" [61] similar to how I felt when looking for examples on Daml, I kept running into the same examples.

5.2.3 Hyperledger Fabric Go

Platform and Development

Since Golang is a GPL it is already widely used and supported In many IDEs , its widely more popular and used for more than smart contracts. As we can see with the trends below Go is the most popular searched and questioned language compared to Daml and Solidity.

When it comes to the Hyperledger API and Go the packages are well supported due to a majority of Hyperledger Fabrics chaincode are written in Go, while they have added other supported GPLs like Java ,Node.JS. Go is typically the first choice and all the newest updates to Hyperledger Fabric are first written in Go as mentioned here [66] [67].

Go is also widely used in creating blockchain systems as it's a compiled language, it has also created the Ethereum Virtual Machine . Many developers choose to use Go as their language over the use of Node.JS due to the fact Go is much faster and lightweight so compilation times are much faster than languages like Node.js [67], these are important factors to consider as compilation speed is important if too slow, fail may occur due to network problems, whereas Go compiles from the source without fetching anything reducing compilation times.

Discussed in the Hyperledger Documentation supporting various GPL languages allowing developers to jump straight into Hyperledger, considering most enterprises will already have the skillset needed to develop smart contracts with no additional training to learn a new language or DSL [68]. One discrepancy I found from my experience creating Hyperledger contracts is that although not needing to learn new languages or hire new developers is true, I found learning how to design smart contracts the most challenging rather than the programming code/syntax itself, similar to what was said with a contract developer on the EthDev Reddit [69] "… Smart Contract programming is something you need to

be familiar with. In a place where code must be perfect, and a very specific niche, those not familiar with smart contracts must learn to be…Go may be slightly easier, its not worth having literally none of the tooling... ", in which I tend to lean more towards from experience in implementing Go chaincode.

I asked on the Hyperledger Reddit about Development with Go and Solidity and the response was tied to the differences between Hyperledger Go and Ethereum Solidity, "I would say that the big difference is not in the language itself, but in the way that the contracts are used. One example is that on the Ethereum blockchain, the contracts are persistent and public…This means that a developer will have to have a thoroughly tested contract before publishing it on the ledger, given that everybody will be able to access it … it will stay there forever. Hyperledger … networks are private…you can determine exactly who has access to your contracts." The user has a point in Hyperledger being private allows developers to not have to worry so much about the tooling and security of a smart contract, although I think tooling from DSL languages not only helps security is makes the contract easier to read and understand making a developer's job easier.

Deploying smart contracts on Hyperledger was harder than expected after following the setup instructions from the documentation and running the test network provided the commands to deploy the contract to the test network took some time to understand as they were long query commands, although the deployment phase was much faster than uploading a contract to Ethereum due to the consensus choices used in Hyperledger there was no consensus wait time unlike Ethereum. Chaincode contracts where pretty much instantaneous as well as invoking and querying. However, after deploying the contract, calling functions were much easier on Ethereum due to the simplicity with Remix, to query and invoke on Hyperledger lengthy chaincode commands were required to read and write to the blockchain examples are see in the Appendix and briefly in the implementation.

Readability and Syntax

When creating the chaincode contracts in Go I found it to be the most difficult to initially read compared to the DSL languages, but this is due to not having the language being specifically design for smart contracts. The language initially looks confusing and longer in comparisons to Solidity and Daml because the language depends on calling the Hyperledger API packages to interact with the blockchain ledger and requiring multiple tests and error checking, resulting in the code becoming much lengthier as presented in the implementation.

Learning the language for the first time took me longer than expected in comparison to Ethereum and Daml, there was much more of a learning curve when it came to development and understanding the capabilities of Go. Learning the Go and its Syntax as well as learning the Hyperledger Fabric API took me a lot longer to understand how to correctly create a smart contract on Hyperledger.

Security

Security concerns with Hyperledger Fabric differ slightly as they are running on private blockchains, the worry about malicious attackers from public are significantly reduced because access to the contract is restricted and requires verification by organizations running the contract.

Smart contracts themselves are prone to code errors and vulnerabilities which can be violated by malicious programmers and developer defects such as coding flaws and design errors are the main reasons that cause smart contract vulnerabilities. There is less of a worry towards vulnerable contracts on Hyperledger due to the network restrictions as well as being able to update chaincode contracts whereas with Ethereum Solidity once the contract is uploaded to the network it is immutable.

Fabrics chaincode were not build with the idea of being strained to DSL unlike Solidity, DSL languages have been designed to implement features to make design flaws and errors less likely. While Fabric tends to use GPL, security vulnerabilities can derive from non-deterministic behaviour, fabric has no currency built into the network and it may not be easy to define how vulnerability it makes the network open to exploits if an attacker gains access to the private blockchain but the cost of an attack attempt if far less in comparison to a public blockchain like Ethereum where Gas limit is involved.

GPL contracts have their advantages but one disadvantage I realised is that while creating contract it requires a lot more thought about not just how the smart contract will execute but how it will be made secure as there is no specific tooling to help with the security of the contract everything needs to be manually done, for example from Solidity I learnt that using “uint” is much more popular and helps security in smart contracts and removes the worry of numbers going negative or overflowing when using“int” (“int” can be still used for handling positive and negatives) using Go I felt overwhelmed and concerned about implementing incorrect code. An in-depth explanation of some of the possible Risks involving Hyperledger Fabric and chaincode was identified from a research paper investigating Hyperledger Fabrics security, they listed numerous possible vulnerabilities when designing contracts, below highlight some [70]:

RISK	EXPLANATION
RANDOM KEY GENERATION	Since, the simulation of the chaincode occurs in each endorsing peer, the random seed for the keys is different. In Go, the random seed is set as 1 and therefore can be easily predicted by a client
OBJECT REIFICATION	The value of the variables is handled through a pointer, which is an address of memory. Therefore, using reified object addresses might cause non-determinism.
GLOBAL STATE VARIABLES	Global state variables that are not stored to the ledger, might change innately and cause inconsistencies to the endorsing results.
RANGE QUERY RISKS	Queries methods to access the Fabric’s state databases and obtain private data, (e.g. the history or the state of a key); are not executed again in the validation phase and can lead to phantom reads, in which the dirty data cannot be detected
CHAINCODE SANDBOXING	Although, Fabric’s chaincode is executed in an isolated docker container and provides sufficient privileges, it can be exploited in a malicious way, such as to execute port scanning, identify and exploit vulnerable peers, install malicious software, and execute attacks.
MAP STRUCTURE ITERATION	Due to the hidden implementation details of the Go programming language, when an iteration with map structure is used, non-determinism may arise, and the order of key values might be different.

Figure 30:Hyperledger Fabric Security Risks from Research paper

Documentation and Resources

I personally found the documentation adequate; I gained a quick understanding on how to setup and develop chaincode and the documentation gave explanations on the different ways Fabric can be setup since the blockchain is so modular and customizable. I did run into a few issues with when trying to implement Hyperledger on Windows, so I switched to Linux and it made things easier and getting started, following the step-by-step instructions felt overwhelming when it came to deploying , querying, and invoking commands on the ledger, although once I got used to using the commands it

become easier to deploy chaincode. I found testing the contract to be more difficult as I was prone to making small errors with the lengthy commands which started to become annoying. Similarly, people discussed the Documentation on Hyperledger Reddit forum [71] , mentioning that they ran into issues trying to use various steps, and following the step-by-step tutorial was problematic at times. Although documentation is subjective to the user, I thought it was a good part to point out my experience and other peoples.

Resources with Hyperledger Fabric had much more support than Daml however less than Solidity when looking for examples of chaincode contracts and setup instructions, I found it difficult to find resources on implementing various projects, I did not run into many issues designing my contract as Hyperledger fabric had a very similar contract example on their GitHub.

When looking at resources for the programming language Go, due to the fact Go is GPL, and it has been around the longest and it has many more use cases then Solidity and Daml, when it came to a programming situation it didn't take long to find a solution. As mentioned, when it came to using the Hyperledger Fabric Go API it took some research to find out how to do the task involving the Hyperledger Fabric API and chaincode, such as making data specific data private. The Golang website had documentation on the Hyperledger packages which became helpful when designing the chaincode and interacting with the blockchain ledger can be found here [72].

5.2.4 Further Comparisons

Smart Contract Readability analysis with law contract research

When analysis total readability of a smart contract while a developer or someone with programming knowledge maybe to determine a smart contract due to familiarity with popular programming languages, non-technical user may find it difficult to determine what the smart contract is doing they may be able to understand some basic contract functionality such as reading the function names. In a business setting this may become problematic when developing agreements between parties as it would highly depend on developers being able to communicate the contracts across to non-technical people.

One online article from Harvard discussing smart contracts within law [73] similar discussed concerns relating to readability issues for non-technical users, they state, "a non-programmer would be at a total loss to understand even the most basic smart contract and is therefore significantly more beholden to an expert to explain what the contract "says" ". This is one primary concern which seems appropriate to mention from the viewpoint of another industry explaining the concerns about smart contracts in law. It's plausible to assume that non-technical users would most likely need to hire an expert like how people hire contract lawyers. Assuming the smart contract is at a complexity level difficult to comprehend one could argue that a non-technical user could read the smart contract like how some nonexpert's can understand some basic contracts in law.

When we analysis the three-contract language they all have similar concerns in regard to contract readability, Daml seems the most appropriate for the case above when it comes to contracts and readability however Daml has some problems and limits to private blockchains as discussed above. Go on the other hand doesn't feel like a suitable contract regarding readability however if expert developers are required then it could be more of a preference towards the developer. If we analyse Solidity for public blockchain use cases while it has better readability than Go but I would argue less readability to Daml, a non-technical user will most likely require expert help in understand the smart contract. While Solidity has to deal with public cases it would be difficult to create a readable smart contract like Daml while keeping the security Ethereum offers.

Below further discusses comparisons on the three languages:

Since smart contracts are public on Ethereum one of the benefits is that there is a much wider community and projects are typically all open sourced to build trust on the blockchain, so various examples of smart contracts are available on websites such as GitHub, developers can learn and take inspiration on and also contribute to better solutions, if we compare this to the Daml and Hyperledger contracts there is much less examples and we can see by analysing trending data in Figure 31Figure 33.

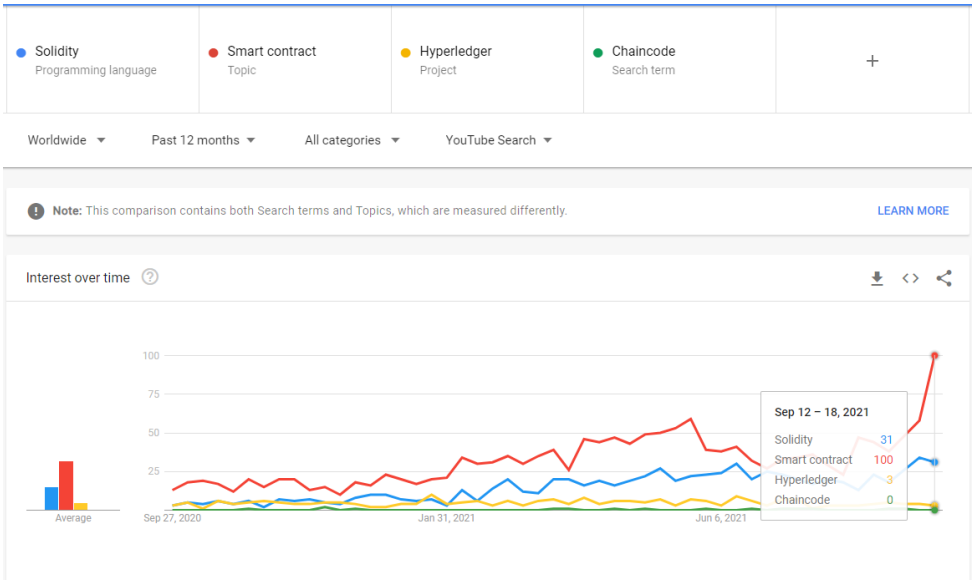


Figure 31:YouTube Trends smart contracts

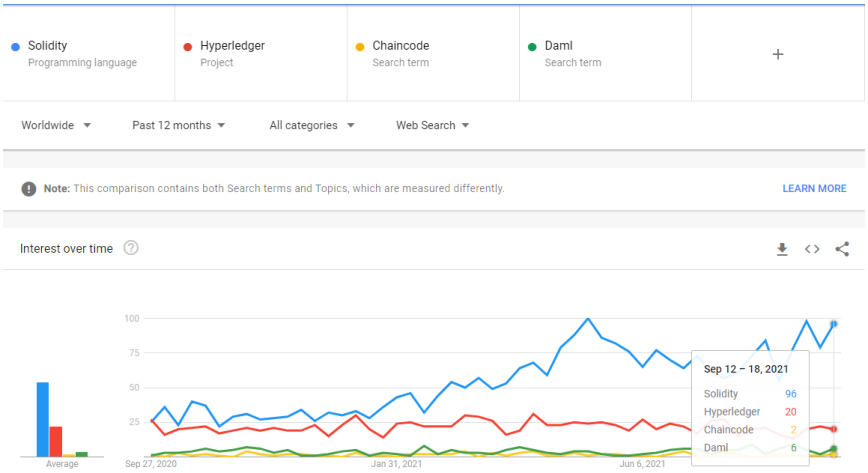


Figure 32: Programming language Google Search

We can both from YouTube, Google Trends [74] and StackOverflow that Solidity's growth has increased proving that there is support around the language. We can also see that Hyperledger has a steady trend rate, this makes similar sense in relations to my findings and research, Hyperledger is focused primarily on enterprise solutions while Ethereum is openly public allowing much more people

to participate in the platform. The reason Go wasn't added to the trending data is due to the fact Go's trends are exaggerated in terms of smart contract programming languages as it's a general-purpose language which means there is far more projects and use cases relating to Go, we can see from the trends on Stackoverflow how popular Go is in comparison to Solidity.

As mentioned in the Hyperledger Fabric discussion, the blockchain uses GPL to help developers jump into designing smart contracts without having to learn a new language, from the trends we can see how popular GPLs can be over DSL languages like Solidity and Daml. Although this doesn't mean they are the best it provides an insight into the popularity Go has over the DSL languages.

We can see below that JavaScript was compared with Go and the difference in questions are hugely in favour with JavaScript. Solidity has been designed to have similar syntax to JavaScript with the same reasoning as Fabric , helping developers jump into coding Solidity contracts easier. While Solidity has specialized features built for smart contracts unlike Go which requires external libraries and more manual coding.

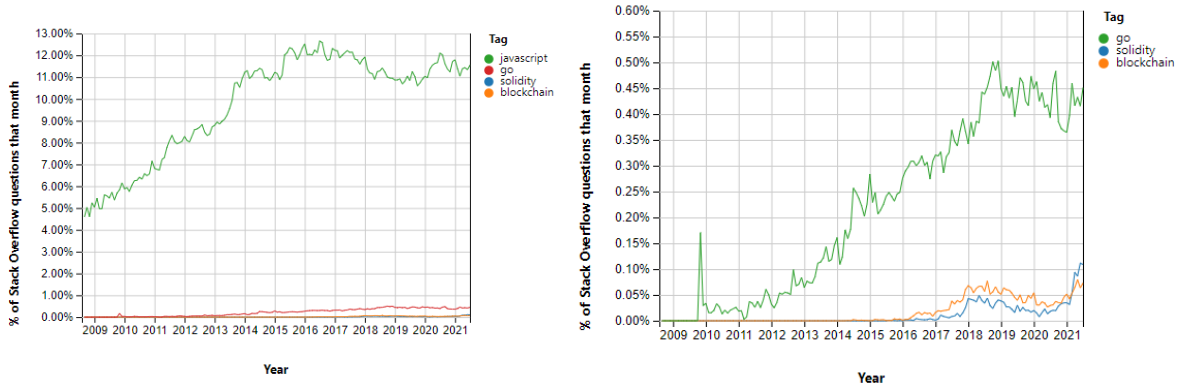


Figure 33:Stackoverflow programming language trends

When we compare the graph on the right, we can see Go significantly higher than Solidity although the case is true because it's a general-purpose language, the stats don't fully indicate the relationships in regards to smart contract programming languages, trends don't determine if it's the best language but it does indicate that there is active development and questions regarding the topic and language. Unfortunately, Daml wasn't represented in the StackOverflow trends [75] as it wasn't an available option, this may be due to it being a highly specific language built for enterprises, we could assume because of the language being targeted for a specific use case as well as focused on enterprises the public development maybe lower since it's a highly specialised language, we can see this with the google trends in Figure 32 .

To conclude the decision between Hyperledger Go and Daml, it heavily relies upon the use cases and needs of the developer, Daml and Hyperledger Go are more focused on enterprise solutions which don't require a currency whereas Solidity is tied down to the Ethereum ecosystem which is publicly available to anyone. In the aspect of business Solidity would be a good choice if the company needs to provide and offer trust or that privacy isn't a problem. On the other hand if we want private smart contracts Go and Daml are great options, I would personally choose Daml over Go for fast development when it comes to financial workflow applications as designing contracts are faster and there is less room to develop vulnerabilities, although this would require some training developers may find it easier to learn as described above with reference to the research found on Daml [63], if a business has experience developers who are comfortable using Go they may be best of sticking to their language of choice is readability isn't a major concern.

After analysing the three languages each language has their benefits and downsides and different use cases, I personally thought Solidity was the best all round language although it does have its limitations in regards to readability and that its tied to the Ethereum ecosystem, when compared against Go I found Solidity a lot easier to understand and creating smart contracts, with its custom features and design it felt easier and the amount of resources available made development for me easier than Go and Daml, although both Go and Solidity where more complex and less readable in comparison to Daml, I thought Daml was too custom and took away the customization offered by Solidity and Go. Although I thought Daml was fantastic for workflow and business logic and the readability Daml offers is great for B2B relations and financial applications. If Daml had more resources available and capabilities for public blockchain I think Daml would be a great alternative to Solidity. When dealing with private blockchains I think Daml should be considered as a replacement to languages like Go and Solidity it brings great features to a blockchain and creating a contract is quick to create as described in the above section.

5.3 Blockchain platforms overview

Blockchain trade-offs play a big factor when having to decide on what blockchain to use or create, from analysis two important areas in blockchain , one for design(Consensus Algorithms) and one for utility (Smart Contracts) we can see the difficulties arise with specific trade-offs.

Taking both smart contract platforms Hyperledger Fabric and Ethereum we see two different types of blockchain designs, one which is tailored towards Enterprises and one tailored publicly for everyone. In the background section we see the discission related to the blockchain trade off trilemma and the trade-offs made between public permissionless and private permissioned blockchains. Through the discussion the various trade-offs have been discussed in relation to consensus mechanism and smart contracts

Hyperledger Fabric is a free highly customizable modular blockchain built for enterprise and tailored for B2B relations. Fabric modularity allows organizations to set how they want their private permissioned blockchain to operate by modifying the consensus, allowing specific organisations to join, and allocating restrictions individuals. While fabric is a permissionless private blockchain, the trade-offs with fabric is the decentralization aspects due to the limitation of the blockchain only certain participants can join. While lowering decentralization we increase the speed by using a less intensive consensus(PBFT), contracts are instantaneous when invoking and querying unlike Ethereum which takes some time to process changes to the blockchain due to their consensus mechanism.

Ethereum offers a publicly available blockchain for anyone to use, while Ethereum offers transparency with its high decentralization and security with its consensus and design, scalability is sacrificed for the other two. A in depth discussion was mentioned throughout the project relating to trade-offs starting primarily at the background section relating to the blockchain trilemma and the Ethereum architecture.

6.0 Evaluation

6.1 Success of the project

Overall, the aims of the project have been successful even with the small changes made throughout the applied part of the project. The research questions from the start of the project have been achieved using the methods of both applied and research, gaining a personal experience with designing and testing to be able to relate to the research found to solidify the results.

Designing smart contracts helped gain some experience and relatability with research, it felt like a success. A lot of learning was achieved from designing smart contracts as problems and issues occurred which wouldn't have been thought about without experience. Gaining experience using Ethereum and Hyperledger helped understand the major differences between public and private blockchains and reasons to why trade-offs are made when designing a blockchain.

I didn't think running the different validators would provide much insight towards understanding how the consensus algorithms work however I gained a lot of information primarily with the differences against the PoST consensus. It helped build a solid evaluation and comparison against the other consensus algorithms as well as research discrepancies.

One downside I found is that I did aim too ambitious at the start of the project, I wanted to created multiple smart contracts and as I started doing this, I wasted a lot of valuable time which could have been aimed at research areas.

6.2 Feedback Implementation

Throughout my project I have received numerous feedbacks from past presentation and report reviews, I have attempt to follow the implementation of the feedback by providing highlights to my own contributions this was done by splitting my dissertation into design, implementation and then discussing my findings against research sources.

I introduced how both smart contracts and consensus algorithms tie into one larger contribution which is the blockchain platform. Detailing the overarching theme between public and private blockchains and the various challenges faced between the technologies, focusing on smart contract technology and consensus algorithms.

I was recommended to show some statistical information to support my conclusions which I provided in the discussion section. As well as data related to hard drive usage and energy consumption to support my conclusions on Chia.

I also added some references to other industry areas, Law in my case to demonstrate the potential problems smart contracts face with readability concerns. Metrics to describe how ease of use is determine was demonstrated by documentation, resources , statistical trends, personal findings.

Conclusions with my research on blockchain platforms are discussed throughout the project primarily in the discussion when referencing smart contract languages.

7.0 Conclusion

7.1 Personal Conclusion

In the thesis, an overview of blockchain and smart technologies were given with details on the architecture of how blockchains are structured with in depth analysis on consensus algorithm and smart contract comparisons. Practical implementations have helped achieve comparisons against research into these technologies evaluating two primary areas within the industry:

- Firstly, blockchain developer who is primarily focused on blockchain implementation and consensus algorithms.
- Secondly, smart contract languages and comparing their features such as readability, ease of use and comparing the platform they are running on and their use cases.

Research Question 1 Trade-offs : A discussion relating to blockchain trade-offs, and the trilemma designers are dealt with, we answered this question within the background research and throughout the experiments and discussion referring to blockchain trade-offs, consensus mechanisms and relations to smart contracts.

Research Question 2 Consensus algorithms : An in-depth discussion into consensus algorithms and comparisons between each of the three (PoW,PoStake,PoST) , describing benefits and limitations in comparison to one another, relations to environmental effects as well as security concerns. While giving my opinion after testing the algorithms and what I think is best with relation to research and findings while providing evidence to my conclusions.

Research Question 3 Smart Contract Comparisons and Analysis : An analysis between Solidity, Daml and Golang made with both research and personal experiments in creating contract in all three languages. The discussion mentioned the differences each language provides and the benefits and limitations they bring, ease of implementation and readability are discussed as well as experiences with personally developing each contract while providing evidence to my conclusions.

7.2 Further Work

Further analysis on Proof of SpaceTime Chia would provide useful after there has some time passed to provide more details and insights into how secure and energy efficient the consensus is in comparison to proof of work and proof of stake.

A lot of my findings were based on personal opinions as the technology is still in infancy there hasn't been much research on smart contract language comparisons. An area to help improve the accuracy of my results would be to perform the same tests on multiple participants from different background which would help identify common strength and weaknesses found by multiple different people. Similar to what Daml does to test their smart contract languages from newly graduated students.

As of writing this dissertation there has been some improvements towards the blockchain trilemma discussed within the background research, new implementations to public blockchains are being introduced to try and solve the three trade-offs by providing sharding and layer 2 solutions to the blockchain to increase scalability while keeping a secure and highly decentralized blockchain. Also, zero knowledge proofs are being discussed to help implement privacy on a public blockchain. This would be an area to consider researching in the upcoming years once the technology has been implemented the theory, it would be interesting to compare the benefits and downsides in comparison to private blockchain like Hyperledger, would this make public blockchains the better solution for everyone if privacy and scalability are solved?

References

[1] M. a. P. S. Valenta, “Valenta, Martin and P. Sandner. “Comparison of Ethereum, Hyperledger Fabric and Corda.,” 2017.

[2] N. Singh, “Ethereum or Hyperledger Fabric?,” 9 2 2018. [Online]. Available: https://medium.com/quillhash/ethereum-or-hyperledger-fabric-259f3c9b8da6. [Accessed 11 9 2021].

[3] S. K. a. H. G. N. Harish, “Distributed Ledger Technology and Blockchain. FinTech Note,” World Bank, Washington, DC, 2017.

[4] V. Kasthala, “Blockchain key characteristics and the conditions to use it as a solution,” 5 7 2019. [Online]. Available: https://medium.com/swlh/blockchain-characteristics-and-its-suitability-as-a-technical-solution-bd65fc2c1ad1. [Accessed 11 9 2021].

[5] V. Buterin, “Why sharding is great: demystifying the technical properties,” 7 4 201. [Online]. Available: https://vitalik.ca/general/2021/04/07/sharding.html. [Accessed 12 9 2021].

[6] “Ethereum Scaling,” 8 8 2021. [Online]. Available: https://ethereum.org/en/developers/docs/scaling/. [Accessed 14 9 2021].

[7] B. Technologies, “medium,” 14 05 2018. [Online]. Available: https://medium.com/@BangBitTech/what-is-consensus-algorithm-in-blockchain-different-types-of-consensus-models-12cce443fc77. [Accessed 9 9 2021].

[8] J.-H. L. Shijie Zhang, “Analysis of the main consensus protocols of blockchain,” *ICT Express,* vol. 6, no. 2, pp. 93-97, 12 08 2019.

[9] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[10] N. T. ,. G. M. ,. F. L. Alfonso Panarello, “Blockchain and IoT Integration: A Systematic Survey,” *Sensors MDPI,* vol. 18, no. 8, pp. 5-37, 6 8 2018.

[11] A. P. S. V. A. Amitai Porat, “Blockchain Consensus: An analysis of Proof-of-Work and its applications.,” 2017.

[12] wackerow, “ethereum.org,” 7 29 2021. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/#:~:text=Proof%2Dof%2Dstake%20is%20the,blocks%20they%20don't%20create.. [Accessed 9 9 2021].

[13] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” Bitcoin.org, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 23 06 2021].

[14] T. &. O. I. Moran, “Simple Proofs of Space-Time and Rational Proofs of Storage,” vol. 11692, no. 1, p. 1, 1 7 2019.

[15] B. Curran, “blockonomi,” 18 04 2020. [Online]. Available: https://blockonomi.com/practical-byzantine-fault-tolerance/. [Accessed 23 06 2021].

[16] I. &. C. V. &. A. R. &. Q. W. &. R. B. Coelho, “Challenges of PBFT-Inspired Consensus for Blockchain and Enhancements over Neo dBFT,” *Future Internet,* vol. 12, no. 8, pp. 1-20, 30 7 2020.

[17] V. Buterin, “Ethereum Whitepaper,” *GitHub repository,* vol. 1, no. 1, pp. 1-23, 2013.

[18] IBM, “What are smart contracts on blockchain?,” IBM, [Online]. Available: https://www.ibm.com/uk-en/topics/smart-contracts. [Accessed 12 9 2021].

[19] H. Fabric, “Membership Service Provider (MSP),” 20. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/membership/membership.html. [Accessed 15 9 2021].

[20] H. Fabric, “Peers,” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/peers/peers.html. [Accessed 15 9 2021].

[21] “Daml - Why Daml?,” [Online]. Available: https://daml.com/why-daml. [Accessed 15 9 2021].

[22] B. Organisation, “Bitcoin Core Requirements and Warnings,” [Online]. Available: https://bitcoin.org/en/bitcoin-core/features/requirements. [Accessed 13 9 2021].

[23] “NiceHash,” [Online]. Available: https://www.nicehash.com/. [Accessed 13 9 2021].

[24] “infura,” [Online]. Available: https://infura.io/. [Accessed 21 9 2021].

[25] Prysm, “Prysm Eth2 Docs,” [Online]. Available: https://docs.prylabs.network/docs/install/install-with-script/. [Accessed 14 9 2021].

[26] n. (. account), “Github (Can not sync pyrmont testnet #9450),” 26 8 2021. [Online]. Available: https://github.com/prysmaticlabs/prysm/issues/9450. [Accessed 27 8 2021].

[27] pyrmont, “beaconcha.in,” [Online]. Available: https://pyrmont.beaconcha.in/validator/0x8c9ff04deef03f90558c3273c540a1ab150f20a25a3944cd24dad7e222d6eb1af8480c19a8e1979b548d17e3eeebe5f4#incomeChart. [Accessed 14 9 2021].

[28] Prylabs, “How to tell if Prysm is running as expected,” [Online]. Available: https://docs.prylabs.network/docs/prysm-usage/is-everything-fine/. [Accessed 14 9 2021].

[29] “Hyperledger Fabric 2.2,” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html. [Accessed 15 9 2021].

[30] “hyperledger github SDK,” [Online]. Available: https://hyperledger.github.io/fabric-chaincode-node/release-2.2/api/. [Accessed 15 9 2021].

[31] “remix ethereum,” [Online]. Available: https://remix.ethereum.org/. [Accessed 16 9 2021].

[32] “Daml Documentation,” [Online]. Available: https://docs.daml.com/getting-started/index.html. [Accessed 16 9 2021].

[33] Alethio, “Are Miners Centralized? A Look into Mining Pools,” 9 5 2018. [Online]. Available: https://media.consensys.net/are-miners-centralized-a-look-into-mining-pools-b594425411dc. [Accessed 2021 9 2021].

[34] “Why is Bitcoin not moving to an ASIC-resistant Proof of Work algorithm?,” 1 2021. [Online]. Available: https://bitcoin.stackexchange.com/questions/101785/why-is-bitcoin-not-moving-to-an-asic-resistant-proof-of-work-algorithm. [Accessed 24 9 2021].

[35] G. Jenkinson, “Ethereum Classic 51% Attack — The Reality of Proof-of-Work,” 10 1 2019. [Online]. Available: https://cointelegraph.com/news/ethereum-classic-51-attack-the-reality-of-proof-of-work. [Accessed 20 9 2021].

[36] N. Carter, “How Much Energy Does Bitcoin Actually Consume?,” 5 5 2021. [Online]. Available: https://hbr.org/2021/05/how-much-energy-does-bitcoin-actually-consume. [Accessed 24 9 2021].

[37] S. L. Y. L. Q. e. a. Jiang, “Policy assessments for the carbon emission flows and sustainability of Bitcoin blockchain operation in China,” *Nat Commun 12,* 2021.

[38] “Cryptocurrency mining and renewable energy: Friend or foe?,” 25 5 2021. [Online]. Available: https://www.smart-energy.com/renewable-energy/cryptocurrency-mining-and-renewable-energy-friend-or-foe/. [Accessed 23 9 2021].

[39] “Eth 2.0 Economics,” [Online]. Available: https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/eth-2.0-economics/. [Accessed 20 9 2021].

[40] “What does the long term future of Ethereum staking look like?,” 2 2021. [Online]. Available: https://www.reddit.com/r/ethstaker/comments/lsuq3f/what_does_the_long_term_future_of_ethereum/. [Accessed 22 9 2021].

[41] “How does POS stake concept deal with rich becoming richer issue?,” 2017. [Online]. Available: https://www.reddit.com/r/ethereum/comments/6x0xv8/how_does_pos_stake_concept_deal_with_rich/. [Accessed 24 9 2021].

[42] “Proof of Stake leads to centralization, with worse consequences than PoW,” 2017. [Online]. Available: https://www.reddit.com/r/ethereum/comments/6d1mca/proof_of_stake_leads_to_centralization_with_worse/. [Accessed 24 9 2021].

[43] “Chia Business Whitepaper,” 1 3 2021. [Online]. Available: https://www.chia.net/assets/Chia-Business-Whitepaper-2021-02-09-v1.0.pdf. [Accessed 20 9 2021].

[44] “Chia,” [Online]. Available: https://www.chia.net/faq/. [Accessed 18 9 2021].

[45] Archilles.H, “How Much Hard Drive Space Do I Really Need - HDD & SSD,” 22 11 2019. [Online]. Available: https://www.shareus.com/computer/how-much-hard-drive-space-do-i-need-hdd-and-ssd.html. [Accessed 20 9 2021].

[46] R. Sharma, “Are Large Mining Pools Bad for Cryptocurrencies?,” 15 8 2021. [Online]. Available: https://www.investopedia.com/tech/are-large-mining-pools-bad-cryptocurrencies/. [Accessed 20 9 2021].

[47] “Miningpoolstats,” [Online]. Available: https://miningpoolstats.stream/chia. [Accessed 20 9 2021].

[48] “chiaforum,” 14 5 2021. [Online]. Available: https://chiaforum.com/t/hpool-51-attack-to-become-100/3217. [Accessed 20 9 2021].

[49] “Reddit - NVME SSD life time is 1-2 weeks?,” 4 2021. [Online]. Available: https://www.reddit.com/r/chia/comments/my8c2s/nvme_ssd_life_time_is_12_weeks/. [Accessed 20 9 2021].

[50] R. Thubron, “Chia farming can reportedly ruin a 512GB SSD in 40 days,” 11 5 2021. [Online]. Available: https://www.techspot.com/news/89626-chia-farming-can-reportedly-destroy-512gb-ssd-40.html. [Accessed 20 9 2021].

[51] J. Hands, “Github Chia-blockchain SSD Endurance,” 16 8 2021. [Online]. Available: https://github.com/Chia-Network/chia-blockchain/wiki/SSD-Endurance. [Accessed 20 9 2021].

[52] chiahodler, “Chia Forum 3% luck, no block from 3 weeks… I surrender, joined HPOOL,” 19 5 2021. [Online]. Available: https://chiaforum.com/t/3-luck-no-block-from-3-weeks-i-surrender-joined-hpool-with-my-few-thousands-plots/4137. [Accessed 20 9 2021].

[53] “Welcome to Remix’s documentation!,” [Online]. Available: https://remix-ide.readthedocs.io/en/latest/. [Accessed 21 9 2021].

[54] R. P. a. M. Siraja, “Why Truffle over Remix Ide,” 6 3 2019. [Online]. Available: https://ethereum.stackexchange.com/questions/67976/why-truffle-over-remix-ide. [Accessed 21 9 2021].

[55] “Solidity Documentation Release 0.8.7,” 11 8 2021. [Online]. Available: https://docs.soliditylang.org/en/v0.8.7/# . [Accessed 21 9 2021].

[56] P. Publishing, “Solidity 101,” 26 1 2019. [Online]. Available: https://medium.com/coinmonks/solidity-101-eec816744e95. [Accessed 21 9 2021].

[57] “Where To Start Learning Solidity That's Beginner-Friendly?,” 2018. [Online]. Available: https://www.reddit.com/r/ethdev/comments/b0dgsg/where_to_start_learning_solidity_thats/. [Accessed 21 9 2021].

[58] B. M. Tomer, “The World of Smart Contracts using Daml & Solidity,” 31 3 2020. [Online]. Available: https://daml.com/blog/engineering/the-world-of-smart-contracts-using-daml-solidity. [Accessed 22 9 2021].

[59] “Some questions about Smart Contract Languages for Masters Thesis,” 6 9 2021. [Online]. Available: https://discuss.daml.com/t/some-questions-about-smart-contract-languages-for-masters-thesis/3077. [Accessed 22 9 2021].

[60] S. Sarraf, “ASX prepares to launch DAML training servic,” 13 2 2020. [Online]. Available: https://www.computerworld.com/article/3527268/asx-prepares-to-launch-daml-training-service.html. [Accessed 22 9 2021].

[61] S. mattaugamer, “Some questions for Masters Thesis in Smart Contract Languages,” 1 9 2021. [Online]. Available: https://www.reddit.com/r/ethdev/comments/pfuh2n/some_questions_for_masters_thesis_in_smart/. [Accessed 23 9 2021].

[62] D. Creer, “How to choose the best platform for your blockchain based system?,” 2 3 2020. [Online]. Available: https://medium.com/gft-engineering/how-to-choose-the-best-platform-for-your-blockchain-based-system-8b9c57862225. [Accessed 22 9 2021].

[63] P. DiMarzio, “How much effort is required to become a Daml-Driven developer?,” 2 10 2018. [Online]. Available: https://daml.com/blog/engineering/how-much-effort-is-required-to-become-a-daml-driven-developer. [Accessed 22 9 2021].

[64] A. Engineering, “Smart Legal Contracts,” 12 2 2020. [Online]. Available: https://daml.com/blog/engineering/smart-legal-contracts. [Accessed 22 9 2021].

[65] D. Assets, “What properties must an enterprise smart contract language have?,” 7 3 2018. [Online]. Available: https://medium.com/daml-driven/what-properties-must-an-enterprise-smart-contract-language-have-b4c251582343. [Accessed 21 9 2021].

[66] ajp, “Does the language in which the hyperledger's chaincode is written matter?,” 13 5 2020. [Online]. Available: https://stackoverflow.com/questions/61730909/does-the-language-in-which-the-hyperledgers-chaincode-is-written-matter. [Accessed 22 9 2021].

[67] I. U. Haq, “Which programming language is best to use for Fabric's chaincode?,” 27 6 2020. [Online]. Available: https://www.researchgate.net/post/Which-programming-language-is-best-to-use-for-Fabrics-chaincode. [Accessed 22 9 2021].

[68] “Hyperledger Fabric Documentation v1.2,” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html. [Accessed 23 9 2021].

[69] mattaugamer, “Some questions for Masters Thesis in Smart Contract Languages,” 1 9 2021. [Online]. Available: https://www.reddit.com/r/ethdev/comments/pfuh2n/some_questions_for_masters_thesis_in_smart/. [Accessed 23 9 2021].

[70] S. a. K. N. a. L. K. a. B. G. a. S. S. Brotsis, “On the Security and Privacy of Hyperledger Fabric:,” 2020.

[71] “Your top challenges building today with Hyperledger Fabric,” 2018. [Online]. Available: https://www.reddit.com/r/hyperledger/comments/arbky9/your_top_challenges_building_today_with/. [Accessed 23 9 2021].

[72] “Hyperledger Fabric Client SDK for Go,” [Online]. Available: https://pkg.go.dev/github.com/hyperledger/fabric-sdk-go. [Accessed 23 9 2021].

[73] S. A. S. M. &. F. L. Stuart D. Levi and Alex B. Lipton, “An Introduction to Smart Contracts and Their Potential and Inherent Limitations,” 26 5 2018. [Online]. Available: https://corpgov.law.harvard.edu/2018/05/26/an-introduction-to-smart-contracts-and-their-potential-and-inherent-limitations/. [Accessed 21 9 2021].

[74] “Google Trends,” [Online]. Available: https://trends.google.com/trends/explore?q=%2Fg%2F11cm045bxr,DAML,%2Fm%2F09gbxjr . [Accessed 21 9 2021].

[75] “StackOverflow,” [Online]. Available: https://insights.stackoverflow.com/trends?tags=solidity%2Cblockchain%2Cgo . [Accessed 21 9 2021].

[76] “Hyperledger Fabric Samples,” [Online]. Available: https://github.com/hyperledger/fabric-samples . [Accessed 24 9 2021].

[77] alexgraham-da, “Asset Transfer,” [Online]. Available: https://github.com/alexgraham-da/daml-asset-custodian/tree/main/daml. [Accessed 22 9 2021].

[78] C. T. a. H. D. T. a. N. D. N. a. N. D. a. N. H. T. a. D. E. Nguyen, “Proof-of-Stake Consensus Mechanisms for Future,Blockchain Networks: Fundamentals,Applications and Opportunities,” *IEEE Access,* vol. 7, pp. 3-6, 2019.

[79] Y. Lau, “Ethereum founder Vitalik Buterin says long-awaited shift to ‘proof-of-stake’ could solve environmental woes,” 27 5 2021. [Online]. Available: https://fortune.com/2021/05/27/ethereum-founder-vitalik-buterin-proof-of-stake-environment-carbon/. [Accessed 18 9 2021].

[80] E. a. D. B. Ghosh, “A Study on the Issue of Blockchain’s Energy Consumption,” pp. 9-14, 1 2021.

[81] K. P. Bram Cohen, “The Chia Network Blockchain,” 9 7 209. [Online]. Available: https://www.chia.net/assets/ChiaGreenPaper.pdf. [Accessed 20 9 2021].

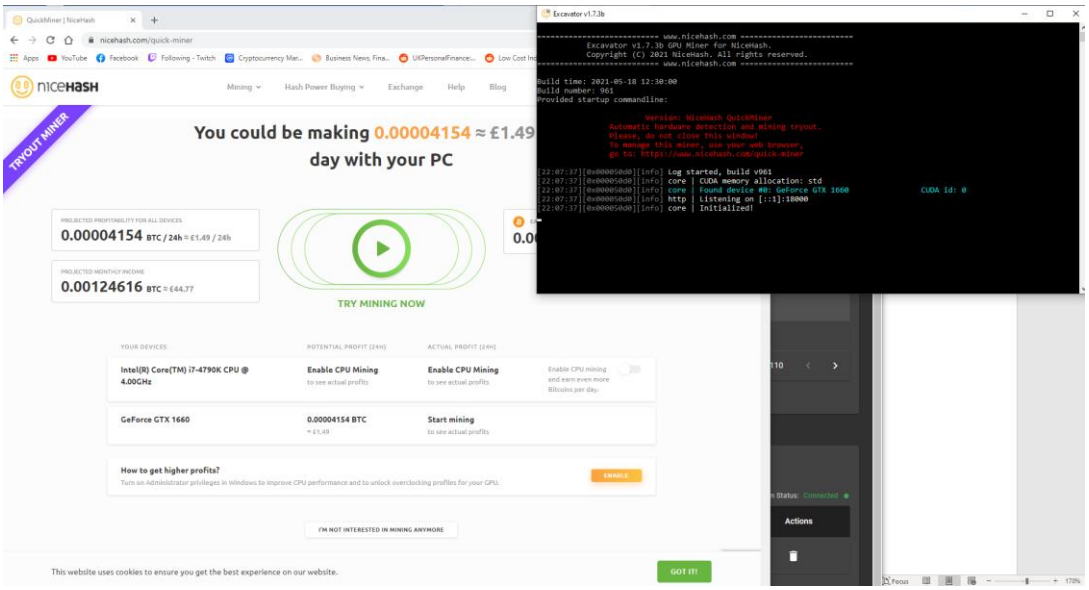
[82] C. Mellor, “Just two weeks after launch, storage-based cryptocurrency Chiacoin drives up disk prices,” 18 3 2021. [Online]. Available: https://blocksandfiles.com/2021/05/18/chia-bitcoin-disk-prices/. [Accessed 20 9 2021].

[83] G. Hoffman, “Chia plotting basic,” 20 2 2021. [Online]. Available: https://www.chia.net/2021/02/22/plotting-basics.html. [Accessed 20 9 2021].

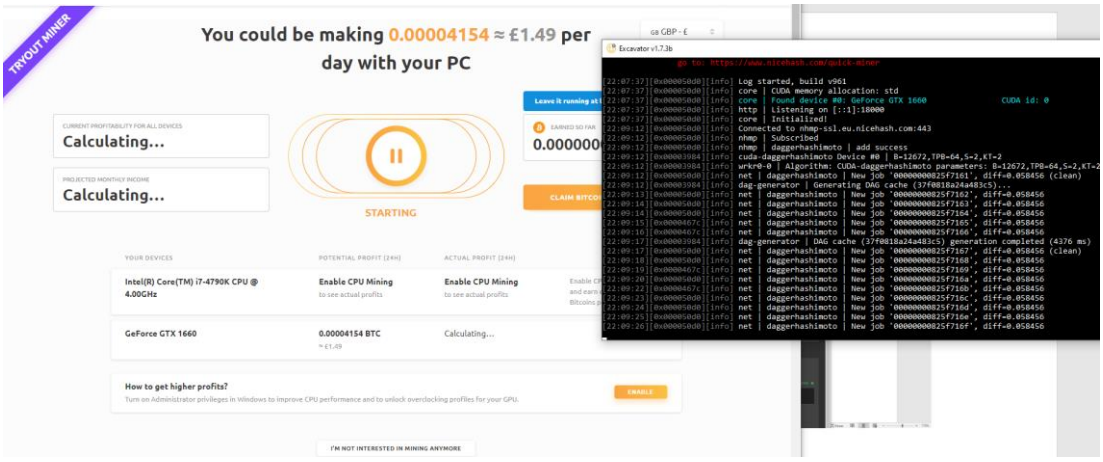
Appendix

Appendix 1: Consensus Algorithms

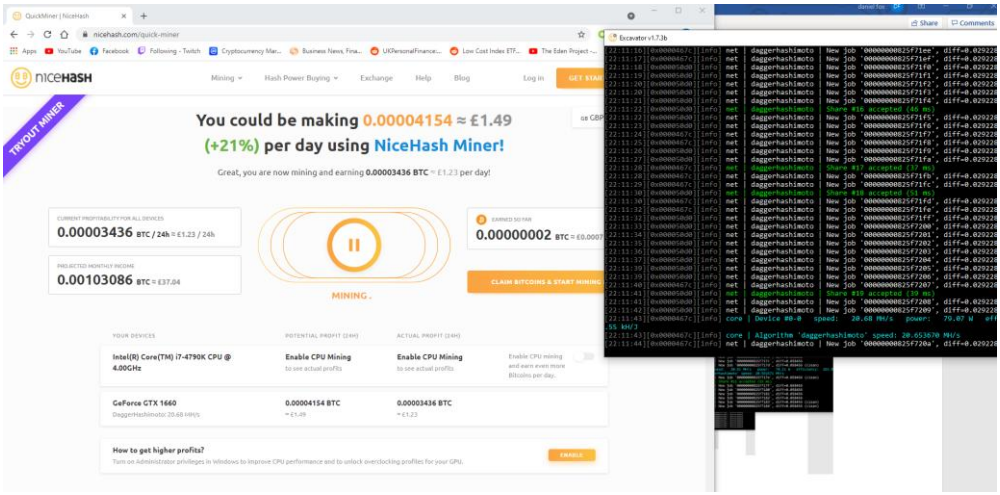
1A: Proof of Work NiceHash Mining Pool Platform



Bitcoin NiceHash QuickStart Mining Platform (Proof of Work)



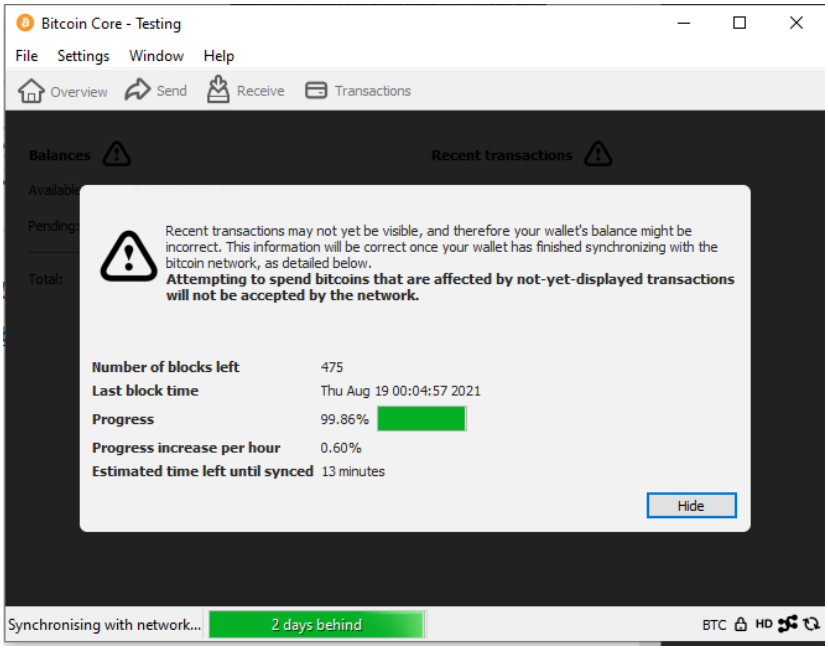
NiceHash QuickStart Init starting Mining process



NiceHash QuickStart Shared Mining Pool Process

1B Bitcoin Core Wallet Mining (Attempt)

The full bitcoin ledger needs to be downloaded first before you can become a mining operator. After this unfortunately the mining didn't work on my computer using the bitcoin core. The only success was through a mining pool like Nice Hash. There is a guide more on mining on the bitcoin website. **https://bitcoin.org/en/full-node#windows-10**

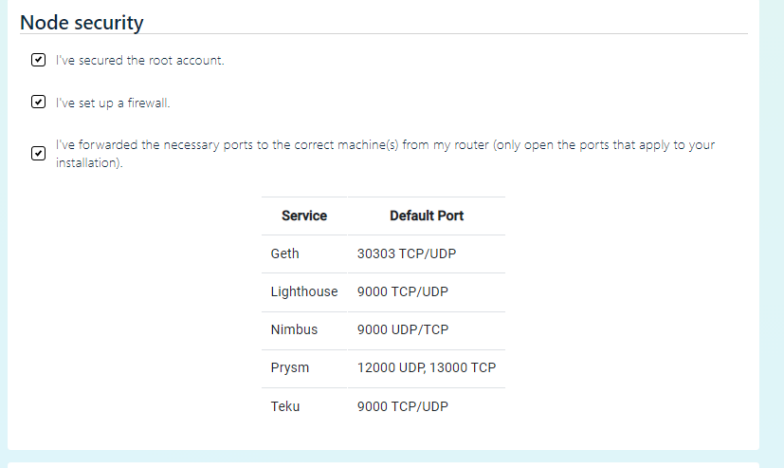


Bitcoin Core Wallet Downloading Ledger

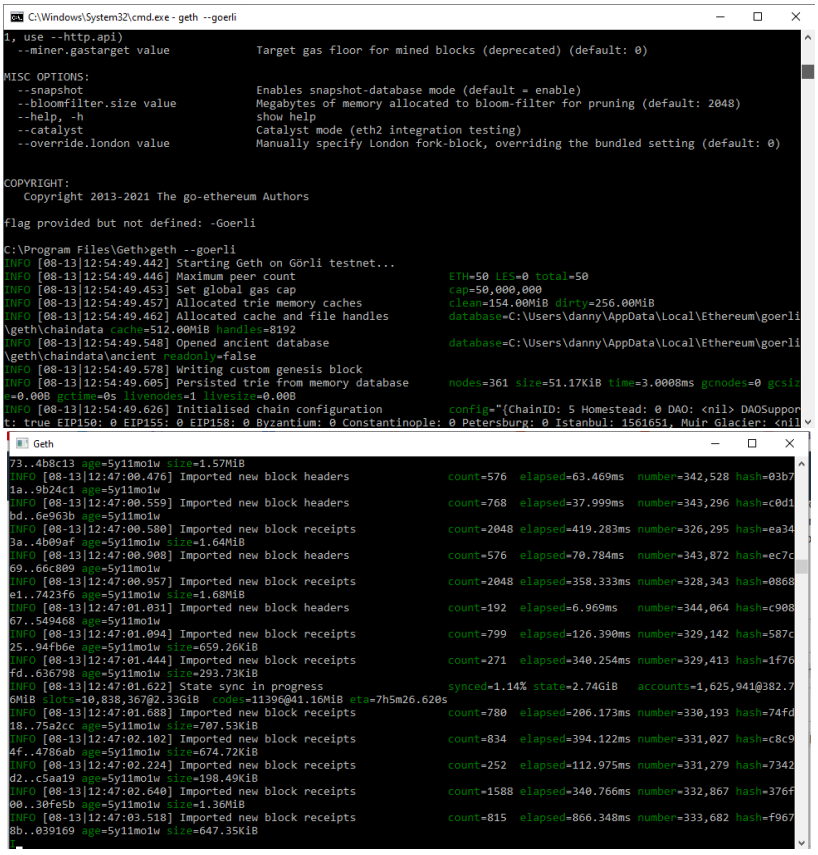
1C: Proof of Stake Implementation process

1.1 Setting up eth1 endpoint (Geth testnet or infura)

Install Geth and run geth -Goerli which is the Ethereum 2.0 testnet pool for proof of stake. See more on this on their eth 2.0 instructions. https://ethereum.org/en/eth2/beacon-chain/ https://docs.prylabs.network/docs/testnet/pyrmont/ .



Validator/Beacon bridge



Running the Go Geth TestNet and downloading ledger

The process took around 7 hours for the ledger on the test net to download, whereas when running the main net this is expected to take more than 7hours since the mainnet ledger has large amounts of records stored.

Run:
```
geth –goerli --http console
```

The above command downloads the ledger, the flag -goerli sets the testnet ledger to download removing this will install the mainnet. –http and console log the outputs.

After Geth installation, start setting up the Prysmlabs beacon to then allow staking and validation. The beacon will communicate to the computer using the RPC and TCP connections and will send out pings occasionally to communicate and update the server. The installation instruction varies depending on OS

Creating wallets and sending eth to staking pool

Sending eth to eth2.0 staking pool by linking wallet address, the download link is provided in the eth2.0 documentation which will give an executable file called eth2deposit-cli-256ea21 this will then further instruct you to setup a eth2 wallet to allow you to send Ethereum from eth1 to an eth2 address. The eth1 wallet used was metamask.

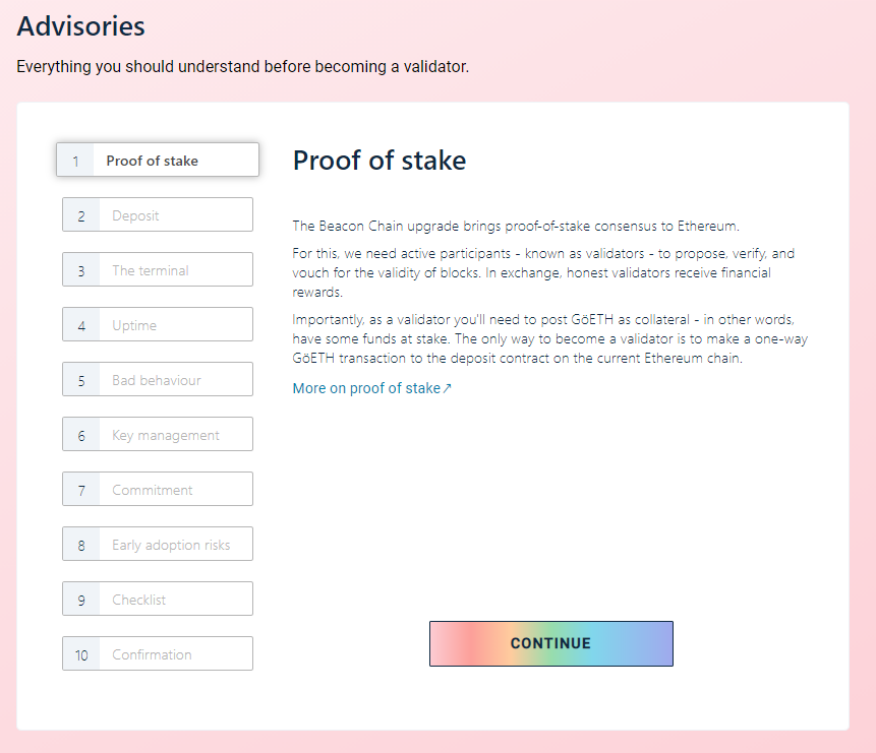
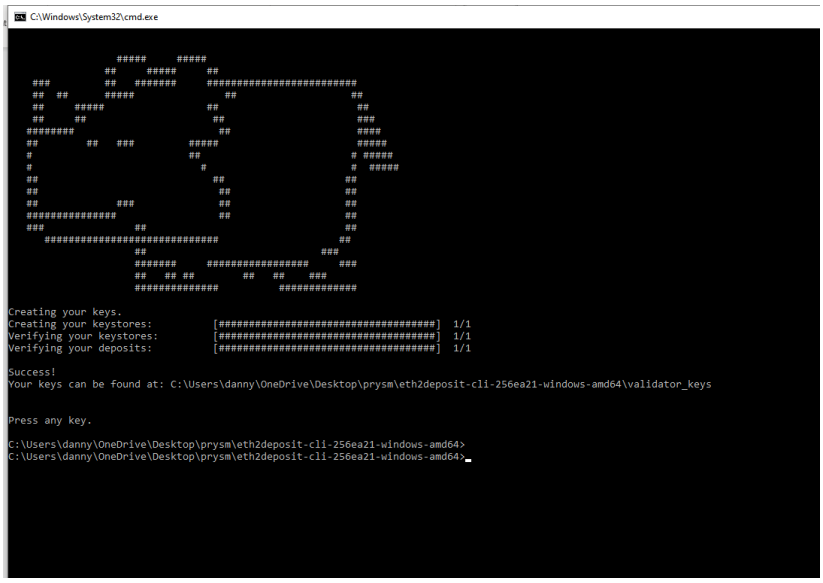


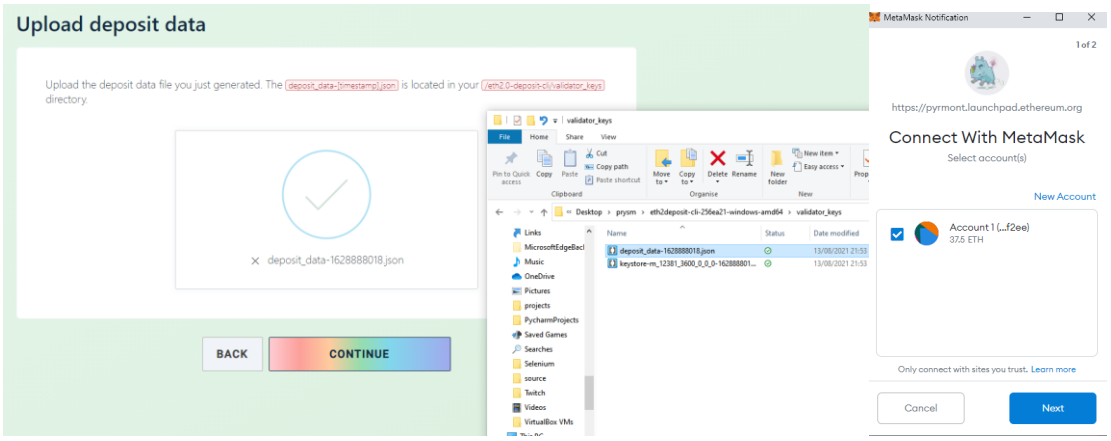
Image of guide used to transfer Metamask wallet to ETH2 wallet

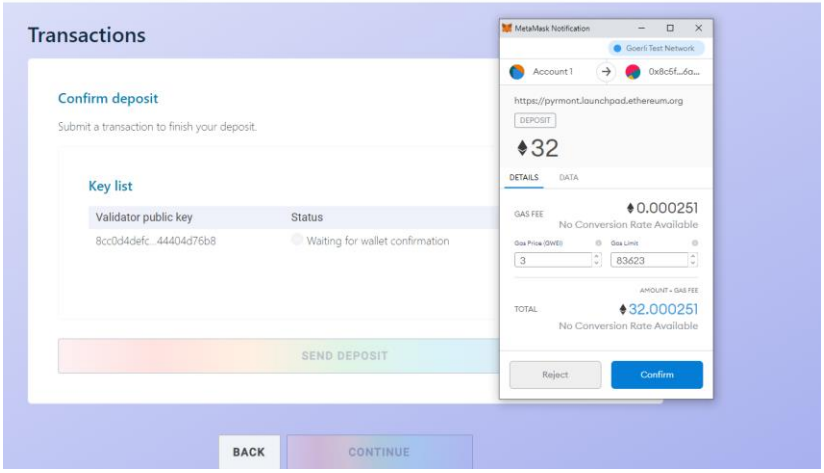
Guide to depositing data and files : https://pyrmont.launchpad.ethereum.org/en/upload-deposit-data



Creating wallet and keys for eth2.0

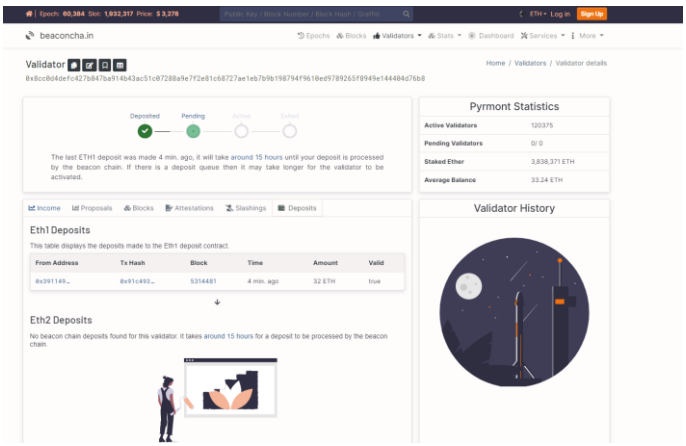
You should see that you have one keystore per validator. This keystore contains your signing key, encrypted with your password. You can use your mnemonic to generate your withdrawal key when you wish to withdraw.





Transfering Eth to ETH2.0 Staking

Once complete you are able to locate the wallet address and look up the transaction on beacon chain to track the process. This was linked to Pyrmont which is related to the prysm eth 2.0 used in this example.



Verify deposit to proof of stake beacon

Recommendation disclaimer

Hardware suggestions are an ever-evolving target. Current minimum requirements are likely to increase by an order of magnitude after the merge and introduction of shard chains. Do your own research before depositing funds.

Hard drive

- To process incoming validator deposits from the Eth1 chain, you'll need to run an Eth1 client as well as your Eth2 client. You can use a third-party service like Infura, but we recommend running your own client to keep the network as decentralised as possible.
- As of February 2021, you'll need ~400GB for the Eth1 mainnet chain data alone (growing at ~1GB/day).
- The Beacon Chain had its genesis on December 01, 2020. It is growing in size over time, and the introduction of sharding will also increase storage, memory, and bandwidth requirements.
- You'll need SSD storage to consistently handle necessary read/write speeds.
- Be sure to account for enough space on your drive until you run maintenance on your node.

CPU and RAM

- Check with client documentation to ensure the hardware you want to use is sufficient and supported.
- Resource usage can vary significantly between clients. Research the different clients if you're working with resource constraints.

Internet

- Ideally your internet connection should be reliable and as close to 24/7 as possible without interruption.
- Ensure your bandwidth can't be throttled and isn't capped so your node stays in sync and will be ready to validate when called.
- You need enough upload bandwidth too. As of February 2021 this is ~700-800 MB/hour, and is likely to increase.

Notes

- Avoid overly-complicated setups and be aware of trade offs. Being offline for brief periods of time will result in small inactivity penalities, but will be recouped easily after being online again for about the same amount of time. Complicated power backups can add to the expense of your setup, and redundant backup validators can lead to slashing. More on slashing risks
- Syncing your Eth1 client may take a few days in the worst-case scenario.

Recommended specs for mainnet

1.3 Starting the beacon chain

Running the command below will start the beacon chain using prysms Ethereum 2.0 client. Below is the local host version which would be used with geth. However due to some issues with the ISP I was currently using it blocked the rpc connection between geth and prysm beacon so it was advised to switch to a server, the choice was infura because its free and popular.

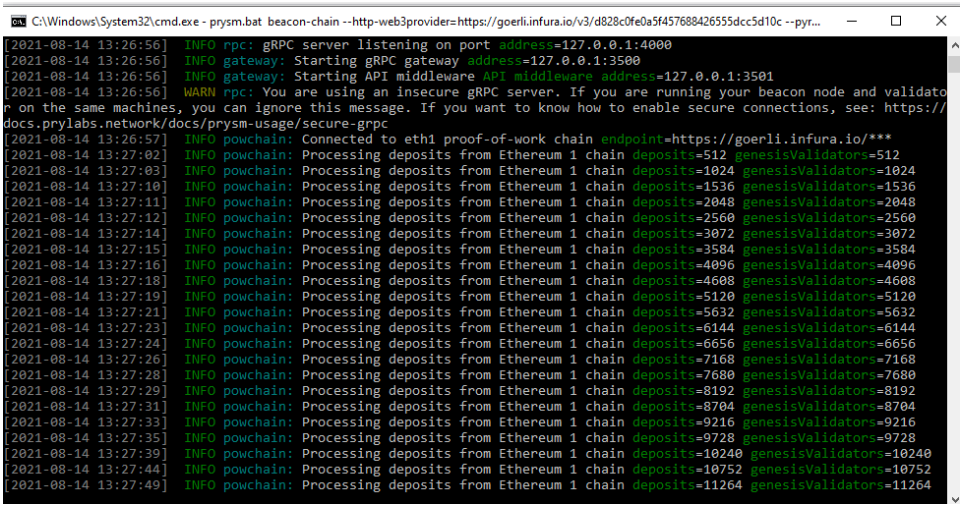
Geth running beacon:

```
prysm.bat beacon-chain --http-web3provider=127.0.0.1:8545 --pyrmont
```

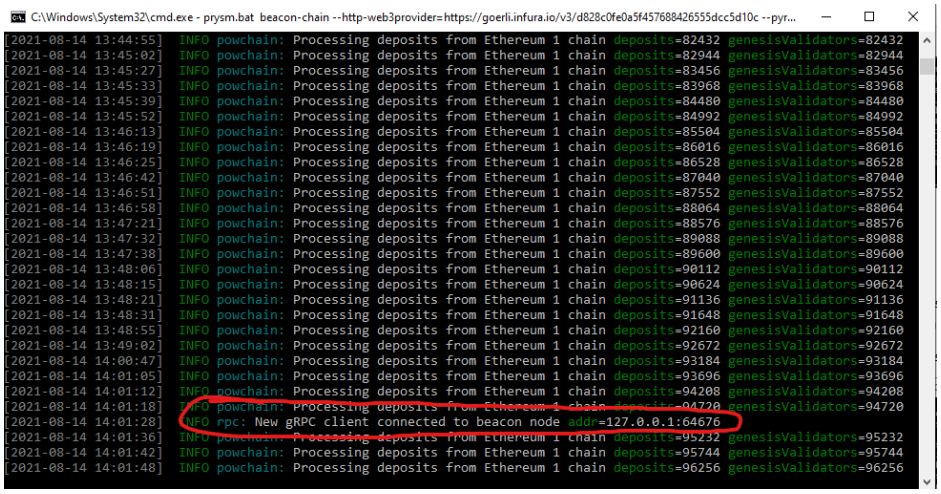
Infura server running beacon:

```
prysm.bat beacon-chain –http-web3provider=https://goerli.infura.io/v3/d828c0fe0a5f457688426555dcc5d10c --pyrmont
```

Note: Error had occurred when running localhost so switched to a server using infura, https://infura.io/dashboard/ethereum/ allowed me to create an eth1 endpoint for the beacon to communicate to.



Beacon install on infura



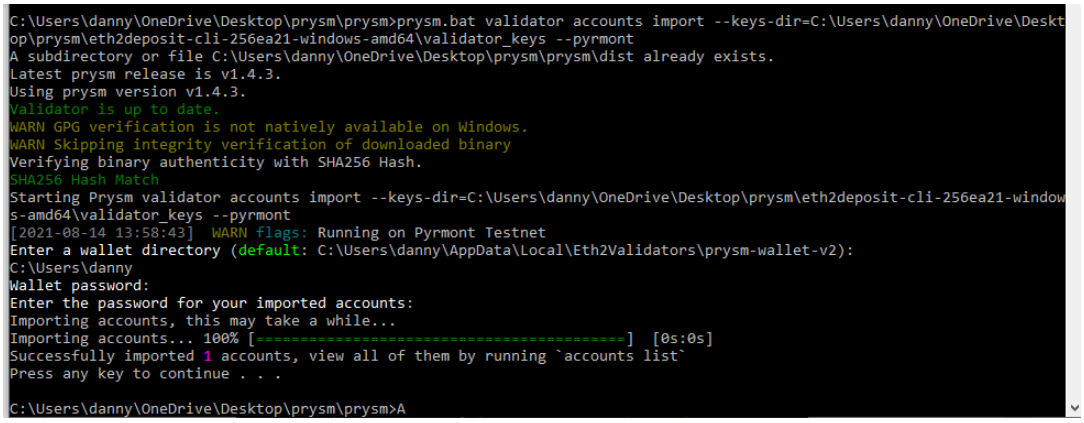
Terminal 2 Beacon chain on infura has connected the validator via RPC

1.4 Running a validator node

Firstly needed to import the accounts and verify the password, this was done with the commands below:

.\deposit.exe new-mnemonic --num_validators 1 --chain pyrmont

prysm.bat validator accounts import --keys-dir=C:\Users\danny\OneDrive\Desktop\prysm\eth2deposit-cli-256ea21-windows-amd64\validator_keys --pyrmont



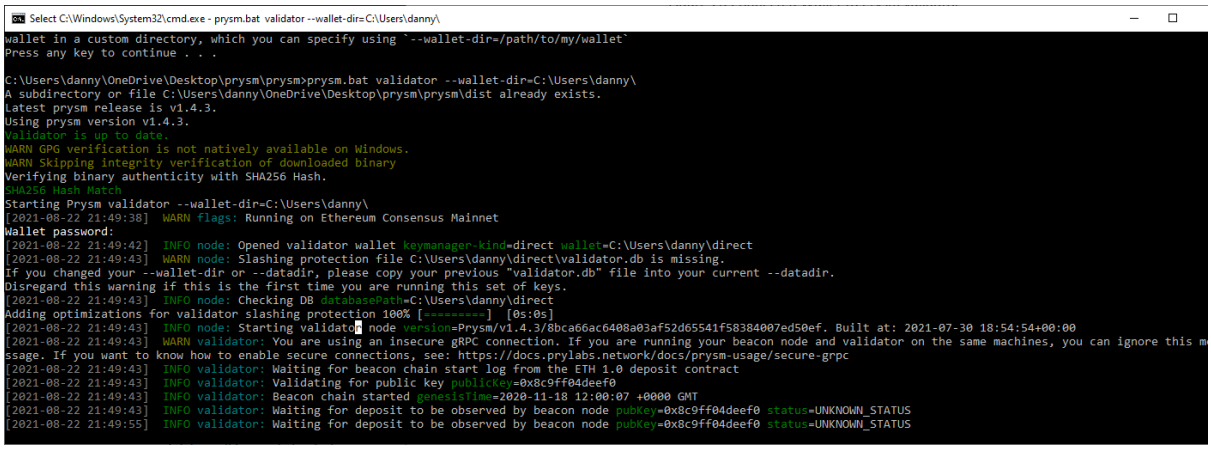
Connected Wallet to Prysm validator

Once run it will prompt for a wallet directory:

e.g: *C:\Users\danny*

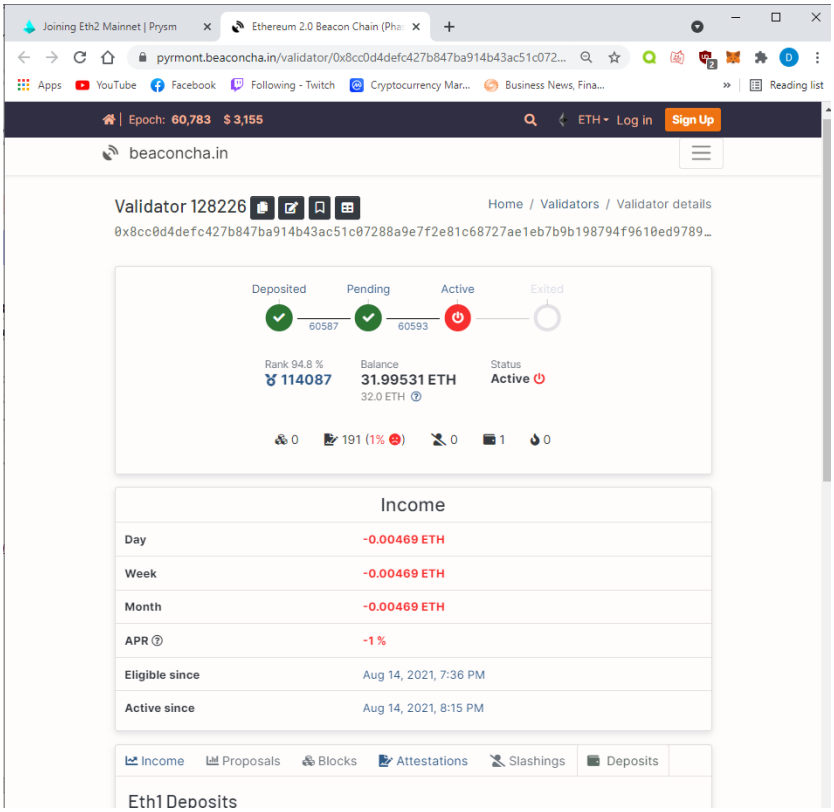
Running validator command:

prysm.bat validator --wallet-dir=C:\Users\danny\ --pyrmont



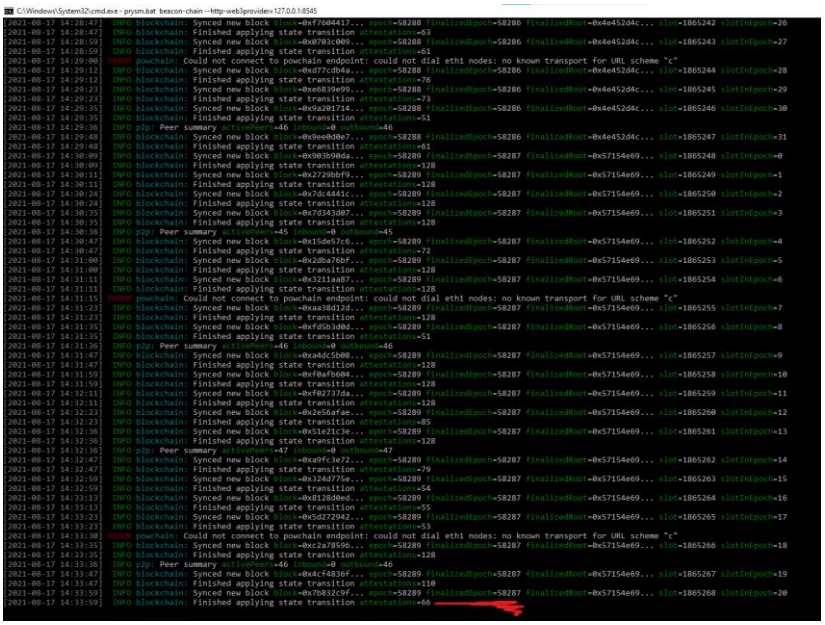
Terminal 1 validator attempting to connect to beacon

If the beacon chain has not synced to the latest block then the validator will prompt a message which states waiting on beacon chain.

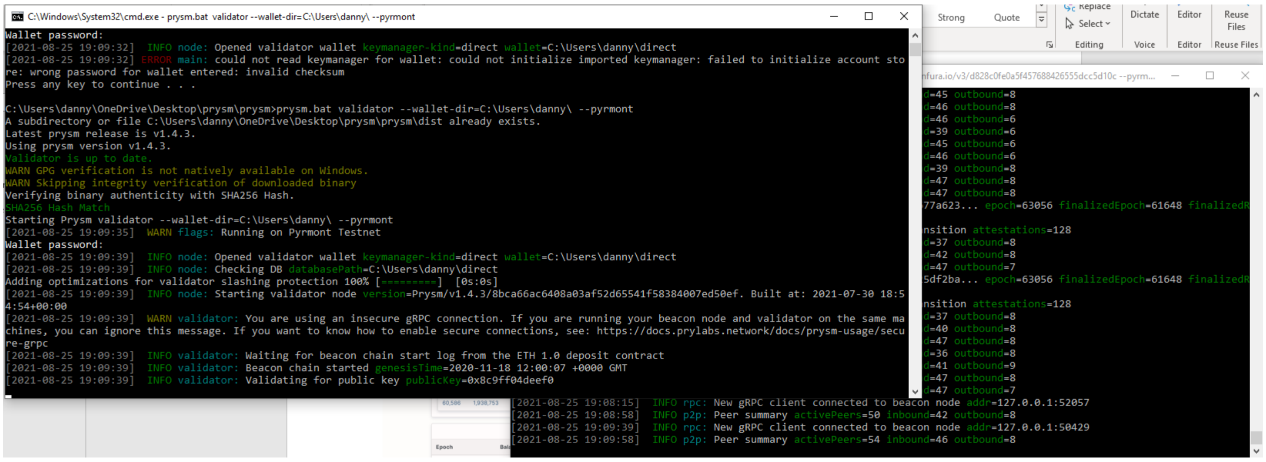


Running Validator

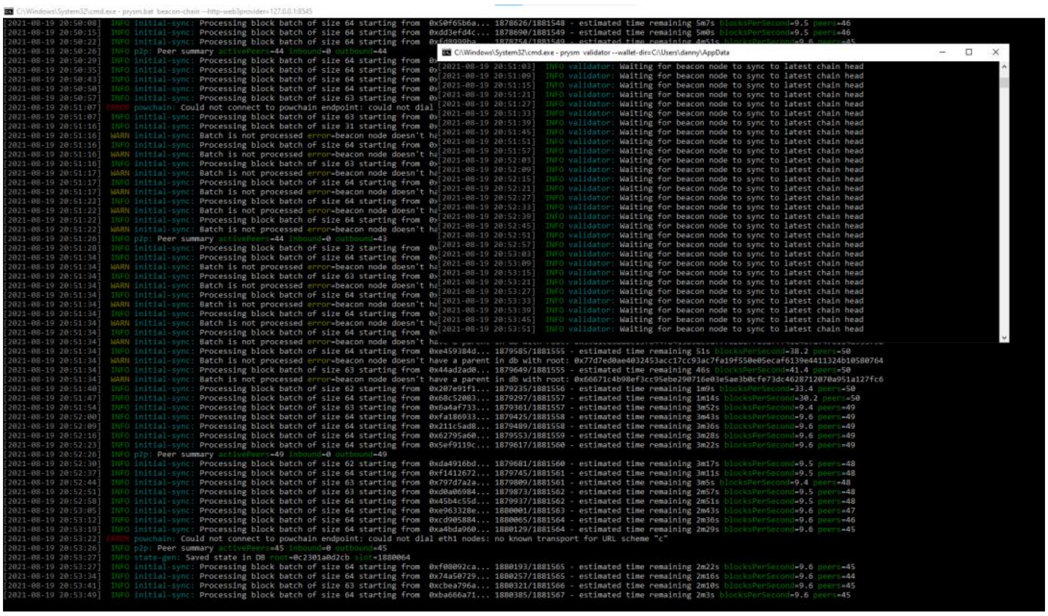
To start validating and earning rewards the beacon has to be rand for a certain period of time before it becomes active, at its shown in the figure above at *60593 epochs* the validator will become eligible and active to earn rewards.



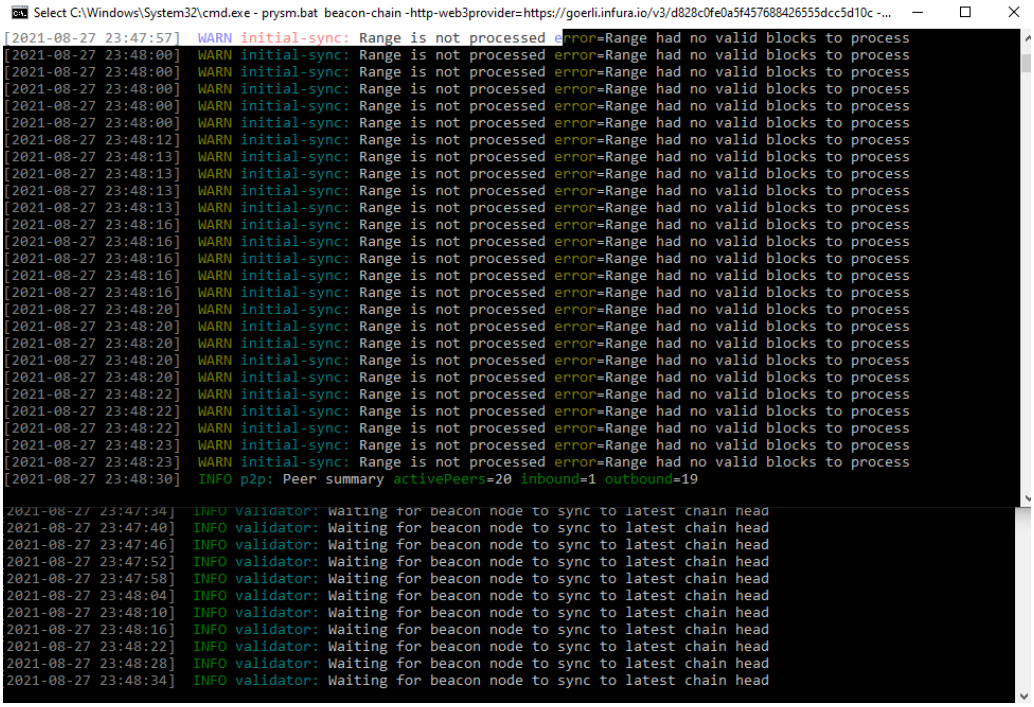
Beacon validator Epochs



Terminal running validators on beacon chain

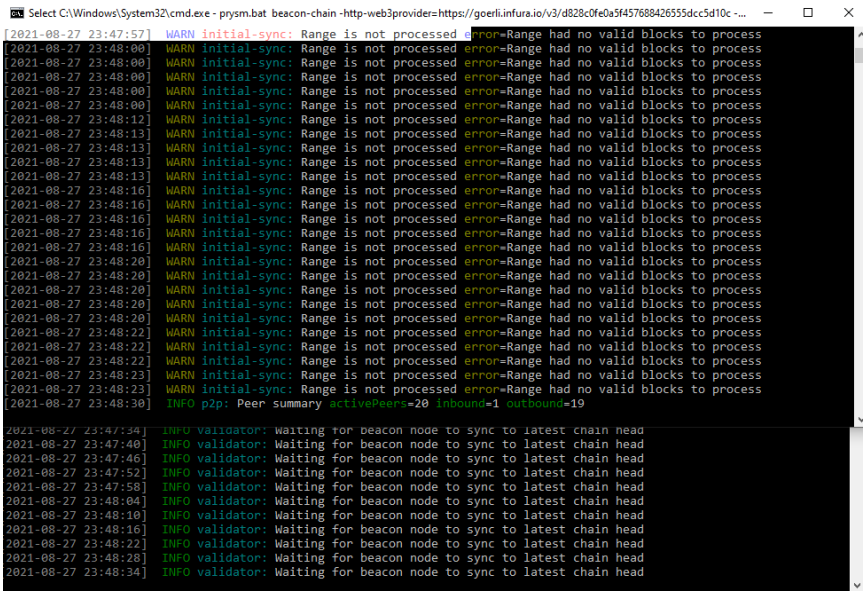


Waiting for beacon to sync for validator node



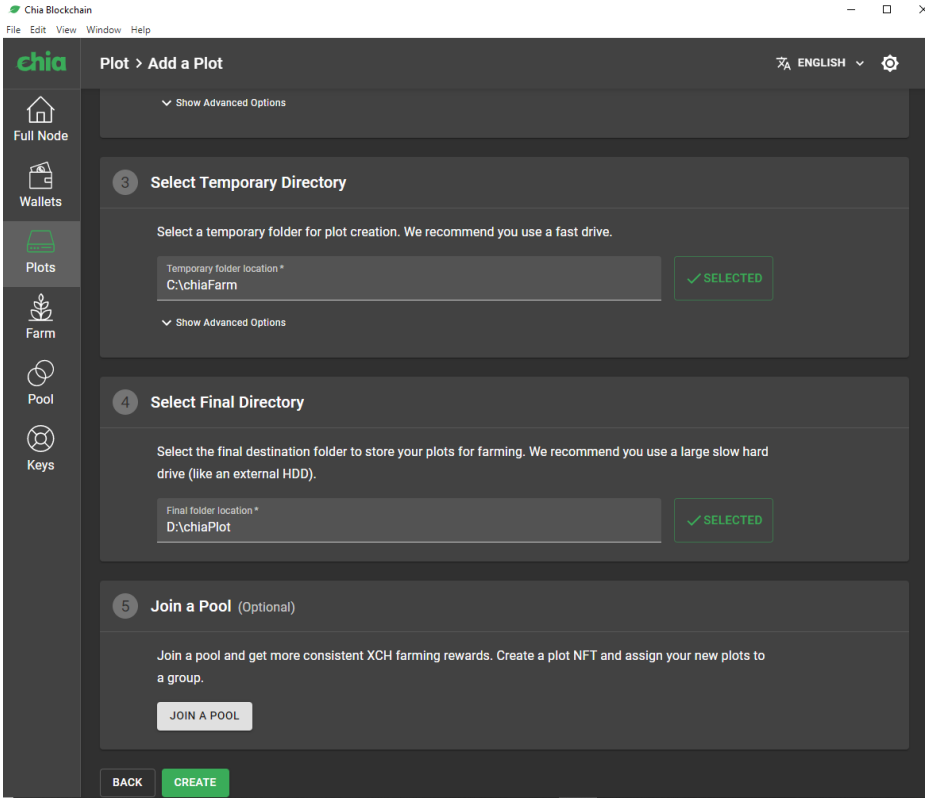
Issue pointed out with validator and beacon at final stage

As discussed in implementation phase an issue was found relating to the Eth2.0 client being used at the time

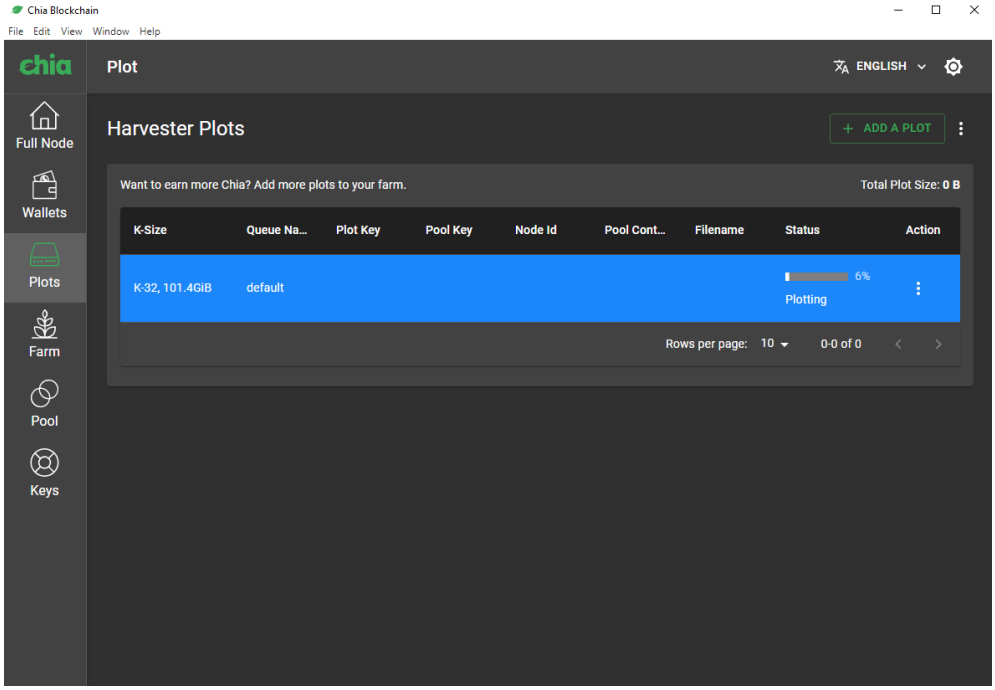


Beacon/Validator Stopped working on Prymont testnet

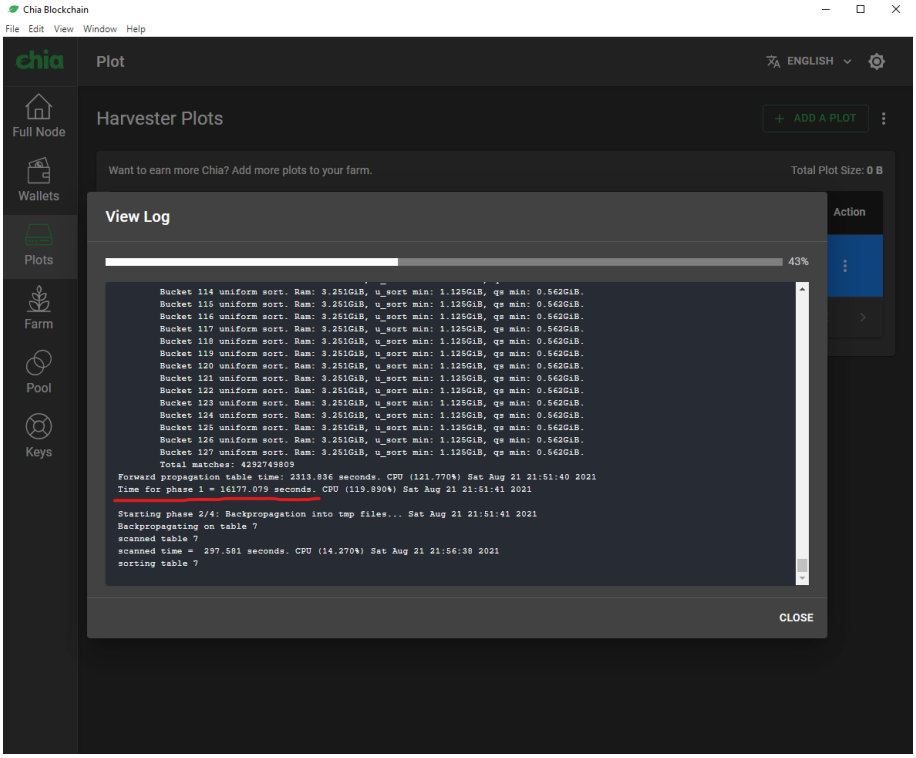
1D: Proof of SpaceTime Implementation process



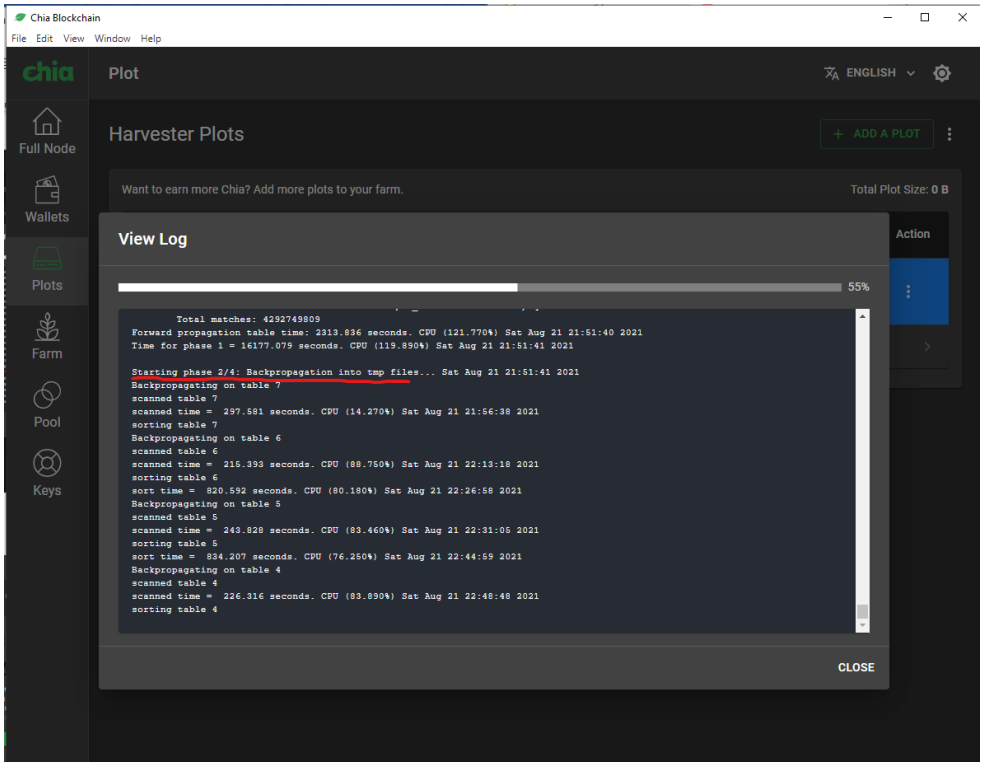
Chia Setting file location for plot and farm



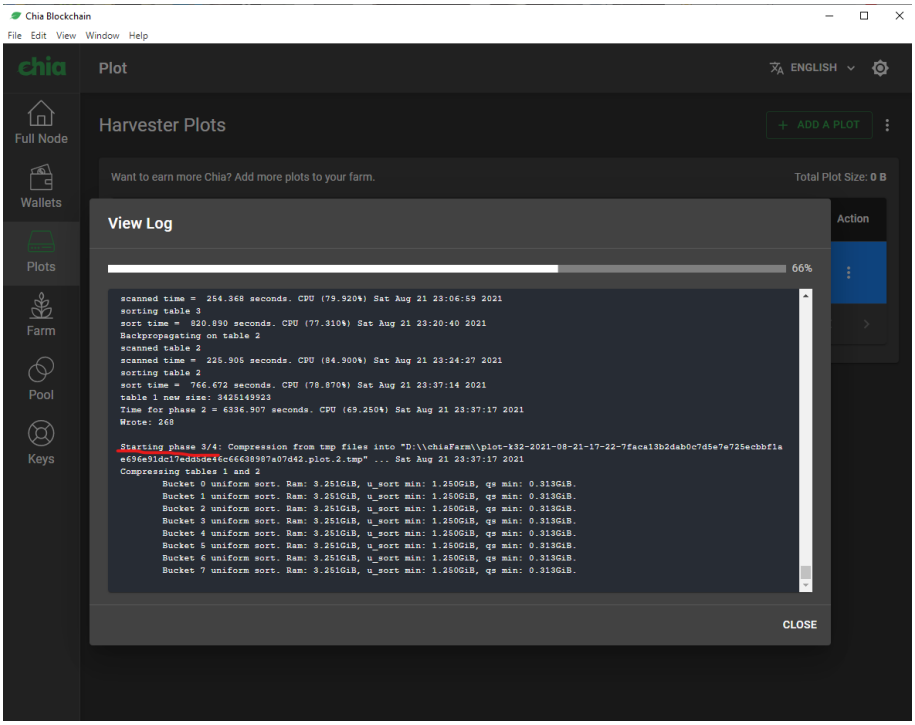
Chia init plot setup



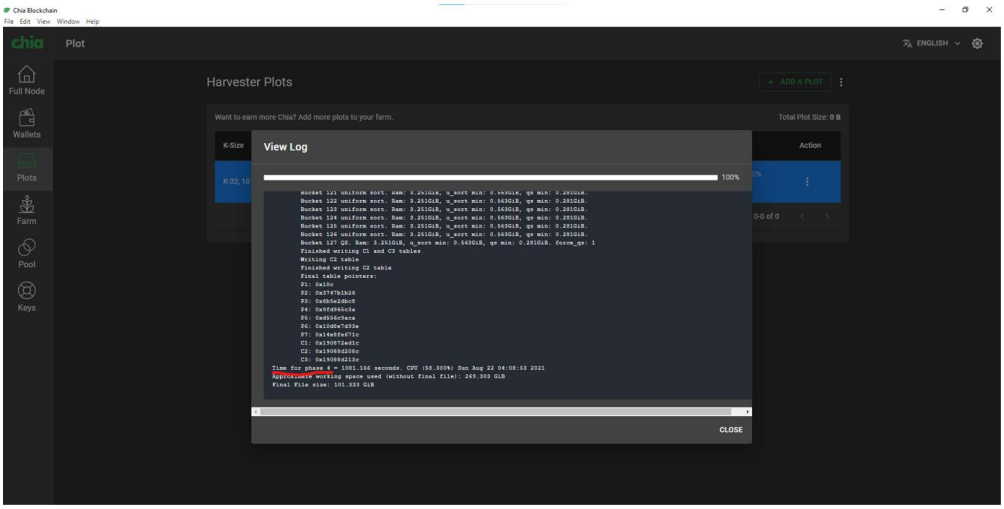
Chia Phase 1 Plotting



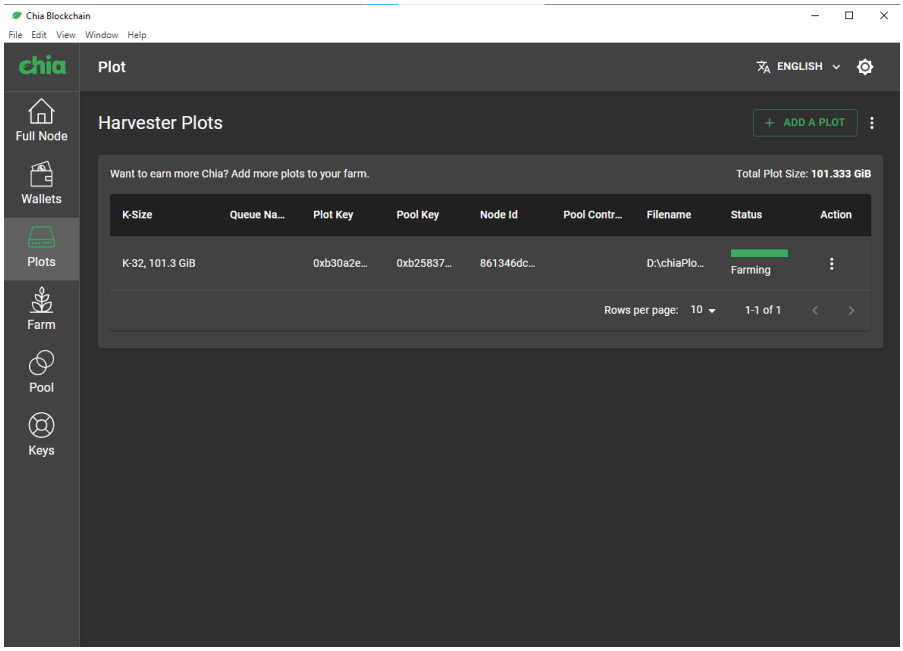
Chia Phase 2 Plotting



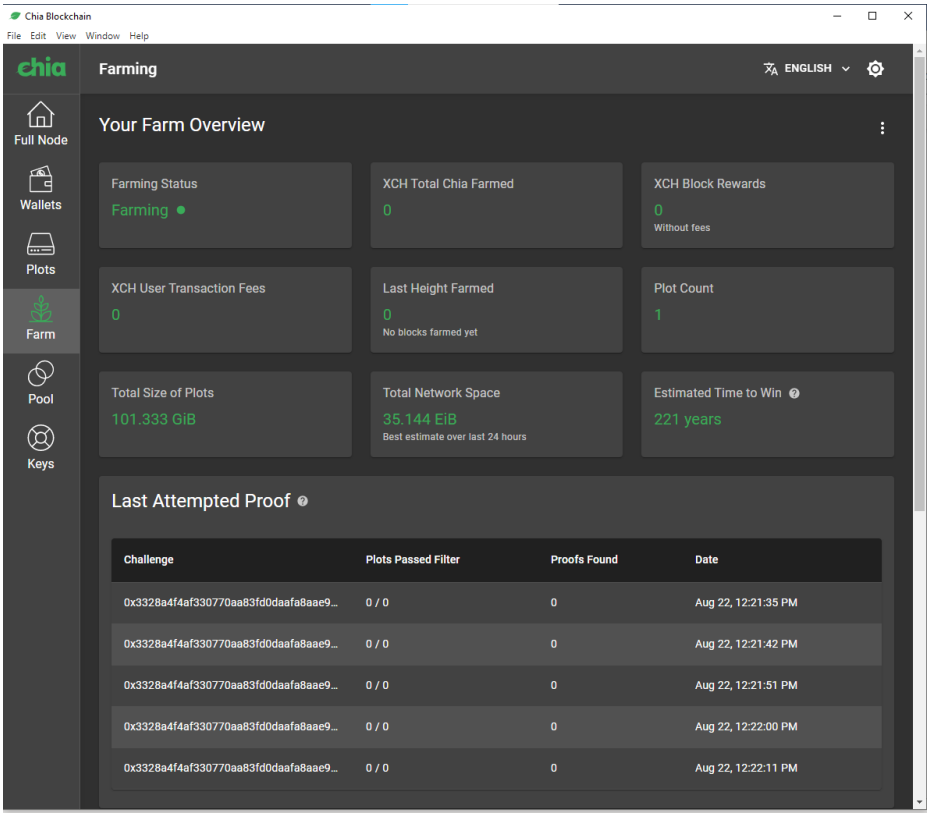
chia Phase 3 Plotting



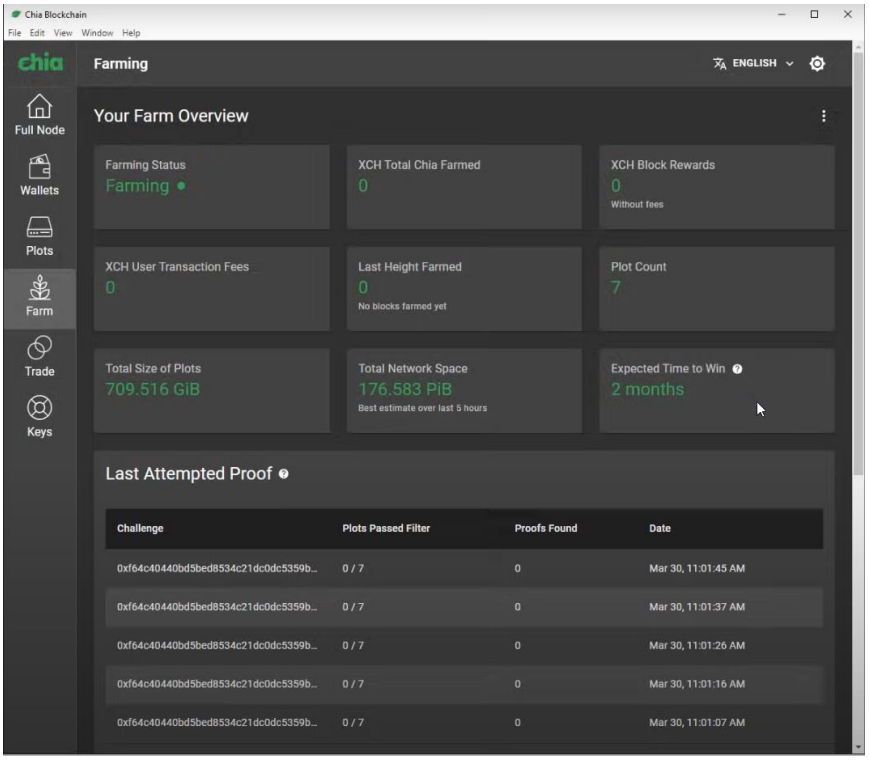
Chia Phase 4 Plotting



Chia Completed Plot Interface



Chia Farming Interface



Example farming rig from YouTube

Appendix 2 : Smart Contracts Programs

2A : Asset Transfer Hyperledger Fabric Golang Chaincode

Asset transfer chaincode with help from Hyperledger Fabric samples [76]

Asset transfer main file

```
package main

import (
    "bytes"
    "crypto/sha256"
    "encoding/json"
    "fmt"
    "log"

    "github.com/hyperledger/fabric-chaincode-go/shim"
    "github.com/hyperledger/fabric-contract-api-go/contractapi" //Provides
the smart contract api interface
)

const (
    sellerPrice = "S"
    bidderPrice = "B"
)

//Init SmartContract
type SmartContract struct {
    contractapi.Contract
}

// Asset details (start with capitals) to work with contract api metadata
/*
e.g
ObjectType: "asset"
AssetID: "asset1"
OwnerOrg: "Org1"
PublicDescription: "This asset is owned by ORG1" / "This asset is for sale"
*/
type Asset struct {
    ObjectType        string `json:"objectType"` // ObjectType is used to
distinguish different object types in the same chaincode namespace
    ID                string `json:"assetID"`
    OwnerOrg          string `json:"ownerOrg"`
    PublicDescription string `json:"publicDescription"`
}

// ***************************  CreateAsset
*****************************************

/*Creates an asset and sets it as owned by the client's org.
The function checks to see if the asset already exist before taking any
action in creating the contract.
Transient data is private to the application-smart contract interaction. It
is not recorded on the ledger and is often used in conjunction with private
data collections
Transient data is confidental and excluded from the ledger.
```

```
Each time the contractapi is passed in a function a transaction context
"ctx" is used, from which you can get the chaincode api functions e.g
GetStub() , GetState() .
*/
func (s *SmartContract) CreateAsset(ctx
contractapi.TransactionContextInterface, assetID, publicDescription string)
error {

    transientMap, err := ctx.GetStub().GetTransient() // Transient data is
private to the application-smart contract interaction.
    if err != nil {
        return fmt.Errorf("error getting transient: %v", err)
    }

    // Must use transient to access Asset properties  as they are private
    privatePropertiesJSON, key := transientMap["asset_properties"]
    if !key {
        return fmt.Errorf("asset_properties key not found in the private
transient map")
    }

    // Verify client id of org and verify it matches peer org id.
    // Client is only authorized to read/write private data from its own
peer for this contract.
    clientOrgID, err := _getClientOrgID(ctx, true) //get the client org id
from transaction context
    if err != nil {
        return fmt.Errorf("failed to get verified OrgID: %v", err)
    }
    //create asset data from struct Asset
    assetCreate := Asset{
        ObjectType:        "asset",
        ID:                assetID,
        OwnerOrg:          clientOrgID,
        PublicDescription: publicDescription,
    }
    assetBytes, err := json.Marshal(assetCreate) // JSON string from a data
structure
    if err != nil {
        return fmt.Errorf("failed to create asset JSON: %v", err)
    }
    //getStub accesses the ledger and requests to update the state to
ledger
    err = ctx.GetStub().PutState(assetCreate.ID, assetBytes) //check and
verify assetCreated.ID
    if err != nil {
        return fmt.Errorf("failed to put asset in public data: %v", err)
    }

    // add private immutable asset properties to owner's private data
collection
    collection := _buildClientOrgName(clientOrgID) //_buildClientOrgName
function passing in ownerOrg: clientOrgID
    //call ledger add private data
    err = ctx.GetStub().PutPrivateData(collection, assetCreate.ID,
privatePropertiesJSON)

    if err != nil {
        return fmt.Errorf("failed to put Asset private details: %v", err)
    }
    return nil
```

```
}

// ****************************** Update Asset
****************************************

/*Update the asset e.g change public description only callable by the
current owner of the asset.
Must verify the ID of the org */

func (s *SmartContract) UpdateAsset(ctx
contractapi.TransactionContextInterface, assetID string, newDescription
string) error {
    // check client org id matches peer org id not needed, use asset
ownership check instead.
    clientOrgID, err := _getClientOrgID(ctx, false)
    if err != nil {
        return fmt.Errorf("failed to get verified OrgID: %v", err)
    }

    assetUpdate, err := s.ReadAsset(ctx, assetID) //Read smartcontract
ledger passing in CTX and assetID to modify data
    if err != nil {
        return fmt.Errorf("failed to get asset: %v , check if exists", err)
    }

    // verify to ensure that client org owns the asset
    if clientOrgID != assetUpdate.OwnerOrg {
        return fmt.Errorf("a client from %s cannot update the description
of a asset owned by %s", clientOrgID, assetUpdate.OwnerOrg)
    }

    assetUpdate.PublicDescription = newDescription     //set new
description
    updatedAssetJSON, err := json.Marshal(assetUpdate) //change json to
string
    if err != nil {
        return fmt.Errorf("failed to marshal asset: %v", err)
    }

    return ctx.GetStub().PutState(assetID, updatedAssetJSON) //update
ledger changing id and updated description
}

// ****************************** Private functions
****************************************
//INSPECTOR AND APPROVAL
func _getClientOrgID(ctx contractapi.TransactionContextInterface, verifyOrg
bool) (string, error) {
    clientOrgID, err := ctx.GetClientIdentity().GetMSPID()
//membershipservice provider ID of organisation e.g {mspid:Org1MSP}
    if err != nil {
        return "", fmt.Errorf("failed getting client's orgID: %v", err)
    }

    if verifyOrg {
        err = _verifyClientOrgMatchesPeerOrg(clientOrgID) //pass into
function to verify client
        if err != nil {
            return "", err
        }
    }
```

```
    return clientOrgID, nil
}

// *Aproval of transactions, assets and pricing *
// _verifyClientOrgMatchesPeerOrg checks the client org id matches the peer
org id.
func _verifyClientOrgMatchesPeerOrg(clientOrgID string) error {
    peerOrgID, err := shim.GetMSPID() //returns the local mspid of the peer
by checking the CORE_PEER_LOCALMSPID env var and returns an error if the
env var is not set
    if err != nil {
        return fmt.Errorf("failed getting peer's orgID: %v", err)
    }

    if clientOrgID != peerOrgID {
        return fmt.Errorf("client from org %s is not authorized to read or
write private data from an org %s peer", clientOrgID, peerOrgID)
    }
    return nil
}

// _setApproval checks that client org currently owns asset and that both
parties have agreed on price
//privatePropertiesJSON makes object unable to change
func _SetApproval(ctx contractapi.TransactionContextInterface, asset
*Asset, privatePropertiesJSON []byte, clientOrgID string, buyerOrgID
string, priceJSON []byte) error {

    // CHECK1: Auth check to ensure that client's org actually owns the
asset

    if clientOrgID != asset.OwnerOrg {
        return fmt.Errorf("a client from %s cannot transfer a asset owned
by %s", clientOrgID, asset.OwnerOrg)
    }

    // CHECK2: Verify that the hash of the passed immutable properties
matches the on-chain hash

    collectionSeller := _buildClientOrgName(clientOrgID)
    setImmutableDataOnChainHash, err :=
ctx.GetStub().GetPrivateDataHash(collectionSeller, asset.ID)
    if err != nil {
        return fmt.Errorf("failed to read asset private properties hash
from seller's collection: %v", err)
    }
    if setImmutableDataOnChainHash == nil {
        return fmt.Errorf("asset private properties hash does not exist:
%s", asset.ID)
    }

    hash := sha256.New()
    hash.Write(privatePropertiesJSON)
    calculatedDataHash := hash.Sum(nil)

    // verify that the hash of the passed immutable properties matches the
on-chain hash
    if !bytes.Equal(setImmutableDataOnChainHash, calculatedDataHash) {
        return fmt.Errorf("hash %x for passed immutable properties %s does
not match on-chain hash %x",
```

```
            calculatedDataHash,
            privatePropertiesJSON,
            setImmutableDataOnChainHash,
        )
    }

    // CHECK3: Verify that seller and buyer agreed on the same price

    // Get sellers asking price
    assetForSaleKey, err := ctx.GetStub().CreateCompositeKey(sellerPrice,
[]string{asset.ID})
    if err != nil {
        return fmt.Errorf("failed to create composite key: %v", err)
    }
    sellerPriceHash, err :=
ctx.GetStub().GetPrivateDataHash(collectionSeller, assetForSaleKey)
    if err != nil {
        return fmt.Errorf("failed to get seller price hash: %v", err)
    }
    if sellerPriceHash == nil {
        return fmt.Errorf("seller price for %s does not exist", asset.ID)
    }

    // Get buyers bid price
    collectionBuyer := _buildClientOrgName(buyerOrgID)
    assetBidKey, err := ctx.GetStub().CreateCompositeKey(bidderPrice,
[]string{asset.ID})
    if err != nil {
        return fmt.Errorf("failed to create composite key: %v", err)
    }
    buyerPriceHash, err :=
ctx.GetStub().GetPrivateDataHash(collectionBuyer, assetBidKey)
    if err != nil {
        return fmt.Errorf("failed to get buyer price hash: %v", err)
    }
    if buyerPriceHash == nil {
        return fmt.Errorf("buyer price for %s does not exist", asset.ID)
    }

    hash = sha256.New()
    hash.Write(priceJSON)
    calculatedPriceHash := hash.Sum(nil)

    // Verify that the hash of the passed price matches the on-chain
sellers price hash
    if !bytes.Equal(calculatedPriceHash, sellerPriceHash) {
        return fmt.Errorf("hash %x for passed price JSON %s does not match
on-chain hash %x, seller hasn't agreed to the passed trade id and price",
            calculatedPriceHash,
            priceJSON,
            sellerPriceHash,
        )
    }

    // Verify that the hash of the passed price matches the on-chain buyer
price hash
    if !bytes.Equal(calculatedPriceHash, buyerPriceHash) {
        return fmt.Errorf("hash %x for passed price JSON %s does not match
on-chain hash %x, buyer hasn't agreed to the passed trade id and price",
            calculatedPriceHash,
            priceJSON,
```

```
            buyerPriceHash,
        )
    }

    return nil
}

//Get clientorg name used to add and verify to private data collection
func _buildClientOrgName(clientOrgID string) string {
    return fmt.Sprintf("_implicit_org_%s", clientOrgID)
}

//Set State
// _SetTransferAssetState performs the public and private state updates for
the transferred asset
//privatePropertiesJSON makes object unable to change
func _SetTransferAssetState(ctx contractapi.TransactionContextInterface,
asset *Asset, privatePropertiesJSON []byte, clientOrgID string, buyerOrgID
string, price int) error {

    asset.OwnerOrg = buyerOrgID              //set the buyerorgid to the
owner in the struct asset
    updatedAsset, err := json.Marshal(asset) //Marshal JSON string from a
data structure which will add to the asset structure.
    if err != nil {
        return err
    }

    err = ctx.GetStub().PutState(asset.ID, updatedAsset) //write state
PutState(ID, updated asset)
    if err != nil {
        return fmt.Errorf("failed to write asset for buyer: %v", err)
    }

    // Transfer the private properties (delete from seller collection,
create in buyer collection)
    collectionSeller := _buildClientOrgName(clientOrgID)
    err = ctx.GetStub().DelPrivateData(collectionSeller, asset.ID)
    if err != nil {
        return fmt.Errorf("failed to delete Asset private details from
seller: %v", err)
    }

    collectionBuyer := _buildClientOrgName(buyerOrgID)
    err = ctx.GetStub().PutPrivateData(collectionBuyer, asset.ID,
privatePropertiesJSON)
    if err != nil {
        return fmt.Errorf("failed to put Asset private properties for
buyer: %v", err)
    }

    // Delete the price records for seller
    assetPriceKey, err := ctx.GetStub().CreateCompositeKey(sellerPrice,
[]string{asset.ID})
    if err != nil {
        return fmt.Errorf("failed to create composite key for seller: %v",
err)
    }

    err = ctx.GetStub().DelPrivateData(collectionSeller, assetPriceKey)
    if err != nil {
```

```
        return fmt.Errorf("failed to delete asset price from implicit
private data collection for seller: %v", err)
    }

    // Delete the price records for buyer
    assetPriceKey, err = ctx.GetStub().CreateCompositeKey(bidderPrice,
[]string{asset.ID})
    if err != nil {
        return fmt.Errorf("failed to create composite key for buyer: %v",
err)
    }

    err = ctx.GetStub().DelPrivateData(collectionBuyer, assetPriceKey)
    if err != nil {
        return fmt.Errorf("failed to delete asset price from implicit
private data collection for buyer: %v", err)
    }

    // Keep record for a 'receipt' in both buyers and sellers private data
collection to record the sale price and date.
    // Add function here ???
    //receiptBuyKey, err :=
ctx.GetStub().CreateCompositeKey(typeAssetBuyReceipt, []string{asset.ID,
ctx.GetStub().GetTxID()})

    return nil
}

// * Transactions and pricing *
// approvePrice adds a bid or ask price to caller's implicit private data
collection
func approvePrice(ctx contractapi.TransactionContextInterface, assetID
string, priceType string) error {
    // client is only authorized to read/write private data from its own
data.
    clientOrgID, err := _getClientOrgID(ctx, true) //verify
    if err != nil {
        return fmt.Errorf("failed to verify OrgID: %v", err)
    }

    transMap, err := ctx.GetStub().GetTransient() //get private data
    if err != nil {
        return fmt.Errorf("error getting transient data: %v", err)
    }

    // Asset price must be retrieved from the transient field as they are
private
    price, ok := transMap["asset_price"]
    if !ok {
        return fmt.Errorf("asset_price key not found transient map")
    }

    collection := _buildClientOrgName(clientOrgID) //get org id

    // set the agreed price in a collection of sub-namespace on priceType
key prefix,
    // Compositekey to avoid collisions between private asset properties,
sell price, and buy price
    assetPriceKey, err := ctx.GetStub().CreateCompositeKey(priceType,
[]string{assetID})
    if err != nil {
```

```
        return fmt.Errorf("failed creating composite key: %v", err)
    }

    // The Price hash will be verified later, the persist price bytes are
passed as is,
    // so there is no risk of nondeterministic marshaling.
    err = ctx.GetStub().PutPrivateData(collection, assetPriceKey, price)
    if err != nil {
        return fmt.Errorf("failed to put asset bid: %v", err)
    }
    return nil
}

// ******************************* AgreeToSell
**************************************

// AgreeToSell adds seller's asking price to seller's private data
//Make sure noone authorised can list the item to sell, only the owner can
func (s *SmartContract) AgreeToSell(ctx
contractapi.TransactionContextInterface, assetID string) error {
    asset, err := s.ReadAsset(ctx, assetID) //read asset from ledger
    if err != nil {
        return err
    }
    //make sure org is verified for payment
    clientOrgID, err := _getClientOrgID(ctx, true)
    if err != nil {
        return fmt.Errorf("failed to get verified OrgID: %v", err)
    }

    // Verify (inspect and aproval) that this clientOrgId actually owns the
asset.
    if clientOrgID != asset.OwnerOrg {
        return fmt.Errorf("a client from %s cannot sell an asset owned by
%s", clientOrgID, asset.OwnerOrg)
    }

    return approvePrice(ctx, assetID, sellerPrice)
}

// ******************************* AgreeToBuy
**************************************

// AgreeToBuy adds buyer's bid price to buyer's private data collection
func (s *SmartContract) AgreeToBuy(ctx
contractapi.TransactionContextInterface, assetID string) error {
    return approvePrice(ctx, assetID, bidderPrice)
}

// SetInspection verifies asset and allows buyer to validate the properties
of
// an asset against the owners private data collection
func (s *SmartContract) SetInspection(ctx
contractapi.TransactionContextInterface, assetID string) (bool, error) {
    transMap, err := ctx.GetStub().GetTransient() //private data
    if err != nil {
        return false, fmt.Errorf("error getting transient: %v", err)
    }

    /// Asset properties retrieved from the transient field as they are
private
```

```
    privatePropertiesJSON, ok := transMap["asset_properties"]
    if !ok {
        return false, fmt.Errorf("asset_properties key not found in the
transient map")
    }

    asset, err := s.ReadAsset(ctx, assetID) //find and read asset from
ledger
    if err != nil {
        return false, fmt.Errorf("failed to get asset: %v", err)
    }

    collectionOwner := _buildClientOrgName(asset.OwnerOrg) //verifty client
org with the asset.ownerOrg from readAsset function
    setImmutableDataOnChainHash, err :=
ctx.GetStub().GetPrivateDataHash(collectionOwner, assetID)
    if err != nil {
        return false, fmt.Errorf("failed to read asset private properties
hash from seller's collection: %v", err)
    }
    //set secure hash e.g the salt tag, so people cant attack and guess the
chaincode asset.
    if setImmutableDataOnChainHash == nil {
        return false, fmt.Errorf("asset private properties hash does not
exist: %s", assetID)
    }

    hash := sha256.New()
    hash.Write(privatePropertiesJSON) //create hash256 for private data
    calculatedDataHash := hash.Sum(nil)

    // verify hash of the passed immutable properties matches the on-chain
hash
    if !bytes.Equal(setImmutableDataOnChainHash, calculatedDataHash) {
        return false, fmt.Errorf("hash %x for passed immutable data %s does
not match on-chain hash %x",
            calculatedDataHash,
            privatePropertiesJSON,
            setImmutableDataOnChainHash,
        )
    }

    return true, nil
}

// ****************************** TransferAsset
**************************************

func getClientImplicitCollectionName(ctx
contractapi.TransactionContextInterface) (string, error) {
    clientOrgID, err := _getClientOrgID(ctx, true)
    if err != nil {
        return "", fmt.Errorf("failed to get verified OrgID: %v", err)
    }

    err = _verifyClientOrgMatchesPeerOrg(clientOrgID)
    if err != nil {
        return "", err
    }

    return _buildClientOrgName(clientOrgID), nil
```

```
}

// TransferAsset checks transfer conditions and then transfers asset state
to buyer.
// TransferAsset can only be called by current owner of the asset
func (s *SmartContract) TransferAsset(ctx
contractapi.TransactionContextInterface, assetID string, buyerOrgID string)
error {
    clientOrgID, err := _getClientOrgID(ctx, false)
    if err != nil {
        return fmt.Errorf("failed to get verified OrgID: %v", err)
    }

    transMap, err := ctx.GetStub().GetTransient() //get private data
    if err != nil {
        return fmt.Errorf("error getting transient data: %v", err)
    }

    privatePropertiesJSON, key := transMap["asset_properties"] //get the
description of asset_properties
    if !key {
        return fmt.Errorf("asset_properties key not found in the transient
map")
    }

    priceJSON, key := transMap["asset_price"] //get price
    if !key {
        return fmt.Errorf("asset_price key not found in the transient map")
    }

    var agreement Agreement                     //make variable based on
agreement struct
    err = json.Unmarshal(priceJSON, &agreement) //string to datastruct
pointer to the agreement variable memory address
    if err != nil {
        return fmt.Errorf("failed to unmarshal price JSON: %v", err)
    }

    asset, err := s.ReadAsset(ctx, assetID) //read data
    if err != nil {
        return fmt.Errorf("failed to get asset: %v", err)
    }

    err = _SetApproval(ctx, asset, privatePropertiesJSON, clientOrgID,
buyerOrgID, priceJSON) //approve
    if err != nil {
        return fmt.Errorf("failed transfer verification: %v", err)
    }

    err = _SetTransferAssetState(ctx, asset, privatePropertiesJSON,
clientOrgID, buyerOrgID, agreement.Price) //set state tp transfer
    if err != nil {
        return fmt.Errorf("failed asset transfer: %v", err)
    }

    return nil
}

func main() {
    //NewChaincode function will error if contracts are invalid e.g. public
functions take in illegal types.
```

```
    //A system contract is added to the chaincode which provides
functionality for getting the metadata of the chaincode.
    chaincode, err := contractapi.NewChaincode(new(SmartContract))
    if err := chaincode.Start(); err != nil {
        log.Panicf("Error starting asset chaincode: %v", err)
    }
    if err != nil {
        log.Panicf("Error create transfer asset chaincode: %v", err)
    }
}
```

Asset transfer Query file

```
package
main

        import (
                "encoding/json"
                "fmt"
                "time"

                "github.com/golang/protobuf/ptypes"
                "github.com/hyperledger/fabric-contract-api-go/contractapi"
        )

        // QueryResult structure used for handling result of query
        type QueryResult struct {
                Record    *Asset
                TxId      string    `json:"txId"`
                Timestamp time.Time `json:"timestamp"`
        }

        type Agreement struct {
                ID      string `json:"asset_id"`
                Price   int    `json:"price"`
                TradeID string `json:"trade_id"`
        }

        // ReadAsset returns the public asset data
        func (s *SmartContract) ReadAsset(ctx contractapi.TransactionContextInterface,
        assetID string) (*Asset, error) {
                // Since only public data is accessed in this function, no access
        control is required
                assetJSON, err := ctx.GetStub().GetState(assetID) //GET ledger data
                if err != nil {
                        return nil, fmt.Errorf("failed to read from world state: %v",
        err)
                }
```

```
	if assetJSON == nil {
		return nil, fmt.Errorf("%s does not exist", assetID)
	}

	var asset *Asset
	err = json.Unmarshal(assetJSON, &asset) //JSON data to go object data
struct
	if err != nil {
		return nil, err
	}
	return asset, nil
}

// GetAssetPrivateProperties returns the immutable asset properties from
owner's private data collection
func (s *SmartContract) GetAssetPrivateProperties(ctx
contractapi.TransactionContextInterface, assetID string) (string, error) {
	// In this scenario, client is only authorized to read/write private
data from its own peer.
	collection, err := getClientImplicitCollectionName(ctx)
	if err != nil {
		return "", err
	}

	privatePropertiesJSON, err := ctx.GetStub().GetPrivateData(collection,
assetID)
	if err != nil {
		return "", fmt.Errorf("failed to read asset private properties
from client org's collection: %v", err)
	}
	if privatePropertiesJSON == nil {
		return "", fmt.Errorf("asset private details does not exist in
client org's collection: %s", assetID)
	}

	return string(privatePropertiesJSON), nil
}

// GetAssetSalesPrice returns the sales price
func (s *SmartContract) GetAssetSalesPrice(ctx
contractapi.TransactionContextInterface, assetID string) (string, error) {
	return getAssetPrice(ctx, assetID, sellerPrice)
}

// GetAssetBidPrice returns the bid price
```

```
func (s *SmartContract) GetAssetBidPrice(ctx
contractapi.TransactionContextInterface, assetID string) (string, error) {
	return getAssetPrice(ctx, assetID, bidderPrice)
}

// getAssetPrice gets the bid or ask price from caller's implicit private data
collection
func getAssetPrice(ctx contractapi.TransactionContextInterface, assetID
string, priceType string) (string, error) {
	collection, err := getClientImplicitCollectionName(ctx)
	if err != nil {
		return "", err
	}

	assetPriceKey, err := ctx.GetStub().CreateCompositeKey(priceType,
[]string{assetID})
	if err != nil {
		return "", fmt.Errorf("failed to create composite key: %v",
err)
	}

	price, err := ctx.GetStub().GetPrivateData(collection, assetPriceKey)
	if err != nil {
		return "", fmt.Errorf("failed to read asset price from implicit
private data collection: %v", err)
	}
	if price == nil {
		return "", fmt.Errorf("asset price does not exist: %s",
assetID)
	}

	return string(price), nil
}

// QueryAssetHistory returns the chain of custody for a asset since issuance
func (s *SmartContract) QueryAssetHistory(ctx
contractapi.TransactionContextInterface, assetID string) ([]QueryResult,
error) {
	resultsIterator, err := ctx.GetStub().GetHistoryForKey(assetID)
	if err != nil {
		return nil, err
	}
	defer resultsIterator.Close()

	var results []QueryResult
	for resultsIterator.HasNext() {
```

```
                        response, err := resultsIterator.Next()
                        if err != nil {
                                return nil, err
                        }

                        var asset *Asset
                        err = json.Unmarshal(response.Value, &asset)
                        if err != nil {
                                return nil, err
                        }

                        timestamp, err := ptypes.Timestamp(response.Timestamp)
                        if err != nil {
                                return nil, err
                        }
                        record := QueryResult{
                                TxId:      response.TxId,
                                Timestamp: timestamp,
                                Record:    asset,
                        }
                        results = append(results, record)
                }

                return results, nil
        }
```

2B Asset transfer Solidity

```
pragma solidity 0.7.0;

contract assetTransfer{

    enum StateType {Active,
    OfferPlaced, PendingInspection,
    Inspected, Apprasied, inspectorApproved,
    BuyerAccepted, SellerAccepted, Accepted,
    Terminated}

    StateType public State;

    address public InitOwner;
    string public Description; //bytes32
    uint public InitPrice;

    address public Buyer;
    uint public OfferPrice;

    address public Inspector;
    address public Apprasor;
```

```
    constructor(uint256 price, string memory detail){
        InitOwner=msg.sender;
        InitPrice=price;
        Description=detail;
        State=StateType.Active;
    }


    function TerminateContract()public{
        if(InitOwner != msg.sender){
            revert();
        }

        State=StateType.Terminated;
    }


    function UpdateAsset(uint256 price, string memory detail) public{
        if (State != StateType.Active || InitOwner != msg.sender){
            revert();
        }
        Description=detail;
        InitPrice = price;
    }

    function MakeOffer(uint256 priceOffer, address inspectorAddress, address a
pprasorAddress)public{
        if(apprasorAddress==address(0)||inspectorAddress==address(0)){//||Init
Owner==msg.sender
            revert();
        }
        if ( State!= StateType.Active){
            revert();
        }
        Buyer=msg.sender;
        Inspector=inspectorAddress;
        Apprasor=apprasorAddress;
        OfferPrice=priceOffer;
        State=StateType.OfferPlaced;

    }

    //Accepted offer await inspection
    function offerAccepted()public{
        if (InitOwner != msg.sender){
```

```
            revert();
        }

        if (State == StateType.OfferPlaced){
            State = StateType.PendingInspection;
        }

    }

    //RejectOffer if offer isnt placed, inspection and approval has been agree
d upon
    function RejectOffer()public{
        if(InitOwner != msg.sender){
            revert();
        }
        //if any of these states arnt set then revert function
        if(State!=StateType.inspectorApproved && State != StateType.Apprasied
        && State != StateType.Inspected && State != StateType.PendingInspectio
n
        && State != StateType.BuyerAccepted && State != StateType.OfferPlaced)
{
            revert();
        }

        Buyer=address(0);
        State = StateType.Active;

    }

    //aceept inspection , approval and owner
    function MakeTransfer()public{
        if( InitOwner != msg.sender && Buyer != msg.sender ){
            revert();
        }
        if(InitOwner == msg.sender
        && State != StateType.inspectorApproved && State != StateType.BuyerAcc
epted){
            revert();
        }
        if (Buyer == msg.sender
        && State != StateType.inspectorApproved && State != StateType.SellerAc
cepted ){
            revert();
        }



        if(Buyer == msg.sender){
```

```
            if(State == StateType.inspectorApproved){
                State = StateType.BuyerAccepted;
            }
            else if (State == StateType.SellerAccepted)
            {
                State = StateType.Accepted;
            }
        }
        else{
            if(State == StateType.inspectorApproved){
                State = StateType.SellerAccepted;
            }
            else if (State == StateType.BuyerAccepted){
                State = StateType.Accepted;
            }
        }

    }
    function ChangeOffer(uint256 priceOffer)public{
        if (State != StateType.OfferPlaced)
        {
            revert();
        }
        if (Buyer != msg.sender)
        {
            revert();
        }

        OfferPrice = priceOffer;
    }

    //revoke offer only if the owner is the person calling the function, reset
 buyer details.
    function revokeOffer()public{
        if (InitOwner != msg.sender){
            revert();
        }
        if(State!=StateType.OfferPlaced && State!=StateType.Inspected
        && State!=StateType.Apprasied && State != StateType.PendingInspection
        && State != StateType.inspectorApproved && State != StateType.SellerAc
cepted){
            revert();
        }
        Buyer=address(0);
        OfferPrice=0;
        State=StateType.Active;
    }
    //setAprraised
```

```
    function setApproval()public{
        //person must be an approver
        if(Apprasor != msg.sender){
            revert();
        }
        if(State == StateType.PendingInspection){
            State = StateType.Apprasied;
        }
        else if(State == StateType.Inspected){
            State = StateType.inspectorApproved;
        }
        else {
            revert();
        }

    }


    //setInspection
    function setInspection()public{

        if (Inspector != msg.sender){
            revert();
        }
        if (State == StateType.inspectorApproved){
            State = StateType.Inspected;
        }
        else if (State == StateType.Apprasied){
            State = StateType.inspectorApproved;
        }
        else{
            revert();
        }
    }

}
```

2C: Asset Transfer Daml

Inspiration from Daml examples and GitHub user alexgraham-da [77]

```
module AssetTransfer where
import DA.List (delete, elemIndex)
type AssetID = ContractId Asset
type CheckAssetsID = ContractId CheckAssets
type MakeProposalID = ContractId MakeProposal
  -- add to Transfer choice of Asset
template Asset
  with
```

```
    owner  : Party
    approver : Party
    name   : Text
    price  : Decimal
    description : Text
  where
    ensure name /= ""
    signatory approver

    controller owner can
      nonconsuming GetPrice: Decimal
        do return price
      Transfer : (MakeProposalID, CheckAssetsID)
        with
          newOwner : Party
          ownerCheckAssets:  CheckAssetsID
          newOwnerAssets:  CheckAssetsID
        do
          updatedSellerInventory <- exercise ownerCheckAssets RemoveAsset with
asset = self
          sale <- create MakeProposal with owner, newOwner, newOwnerAssets, ..
          return (sale, updatedSellerInventory)

    controller owner can
      GetAssetName: Text
        do return name

      Adjustprice: AssetID
        with
          newprice: Optional Decimal
        do
          case newprice of
            Some y -> create this with price = y
            None -> create this

      ChangeAssetName: AssetID
        with
            newAssetName: Text
        do
          create this
            with
              name = newAssetName

    nonconsuming choice NonControllerGetAssetName: Text
      with actor: Party
      controller actor
        do return name
```

```
template CheckAssets
  with
    owner: Party
    assets: [AssetID]
  where
    signatory owner

    controller owner can
      -- check if assetId is already added
      nonconsuming CheckIfAdded: Bool
        with
          asset: AssetID
        do
         case elemIndex asset assets of
          None ->  return False
          Some y -> return True

      -- create a new contract with added asset
      AddAsset: CheckAssetsID
        with
          asset: AssetID
        do
          create this with assets = asset :: assets

      -- create a new contract with removed asset
      RemoveAsset: CheckAssetsID
        with
          asset: AssetID
        do
          let newAssets = delete asset assets
          create this with
            assets = newAssets
template Operator
  with
    approver: Party
    owner: Party
    newOwner: Party
  where
    signatory approver, owner, newOwner

    controller approver can
      UpdateInventories: (CheckAssetsID,CheckAssetsID)
        with
          asset: ContractId Asset
          ownerCheckAssets: CheckAssetsID
          newOwnerAssets: CheckAssetsID
```

```
        do

          updatedSellerInventory <- exercise ownerCheckAssets RemoveAsset with
 asset = asset

          updatedBuyerInventory <- exercise newOwnerAssets AddAsset with asset
 = asset

          return (updatedSellerInventory, updatedBuyerInventory)


template MakeProposal
  with
    owner: Party
    newOwner: Party
    newOwnerAssets:  CheckAssetsID
    approver : Party
    name   : Text
    price  : Decimal
    description : Text
  where
    signatory owner

    controller newOwner can
      AcceptOffer: (AssetID, CheckAssetsID)
        do
          product <- create Asset with
            owner = newOwner
            approver = approver
            name = name
            price = price
            description = description

          updatedBuyerInventory <- exercise newOwnerAssets AddAsset with asset
 = product

          return (product, updatedBuyerInventory)
```

Script and testing file

```
module Script where
import Daml.Script
import DA.Assert
import AssetTransfer
-- Testing
-- setup : Script()

setup : Script ()
```

```
setup = script do
    alice <- allocatePartyWithHint "Alice" (PartyIdHint "Alice")
    bob <- allocatePartyWithHint "Bob" (PartyIdHint "Bob")

    aliceItem <- submit alice do
      createCmd Asset with
        approver = alice
        owner = alice
        name = "Laptop"
        price = 1000.00
        description = "Electronics"

    aliceInitCheckAssets <- submit alice do
      createCmd CheckAssets with
        owner = alice
        assets = []

    bobInitCheckAssets <- submit bob do
      createCmd CheckAssets with
        owner = bob
        assets = []

    (aliceProposalToBob, aliceCurrentCheckAssets) <- submit alice do
      exerciseCmd aliceItem Transfer with newOwner = bob, ownerCheckAssets = a
liceInitCheckAssets, newOwnerAssets = bobInitCheckAssets


    (bobItem, bobNewCheckAssets) <- submit bob do
      exerciseCmd aliceProposalToBob AcceptOffer


    name <- submit bob do
      exerciseCmd bobItem NonControllerGetAssetName with actor = bob

    name === "Laptop"

    -- create new contract with name === 'Monitor'
    newAsset <- submit bob do
      exerciseCmd bobItem ChangeAssetName with newAssetName = "Monitor"

    -- getname
    newAssetName <- submit bob do
      exerciseCmd newAsset NonControllerGetAssetName with actor = bob

    -- check if matches
    newAssetName === "Monitor"

    adjustedAsset <- submit bob do
      exerciseCmd newAsset Adjustprice with newprice = Some 200.00
```

```
    newprice <- submit bob do
      exerciseCmd adjustedAsset GetPrice

    newprice === 200.00

    emptyInitCheckAssets <- submit alice do
      createCmd CheckAssets with
        owner = alice
        assets = []

    verifyEmpty <- submit alice do
      exerciseCmd emptyInitCheckAssets CheckIfAdded with asset = adjustedAsset

    verifyEmpty === False

    inventoryWithAsset <- submit alice do
      exerciseCmd emptyInitCheckAssets AddAsset with asset = adjustedAsset

    verifyNotEmpty <- submit alice do
      exerciseCmd inventoryWithAsset CheckIfAdded with asset = adjustedAsset

    verifyNotEmpty === True

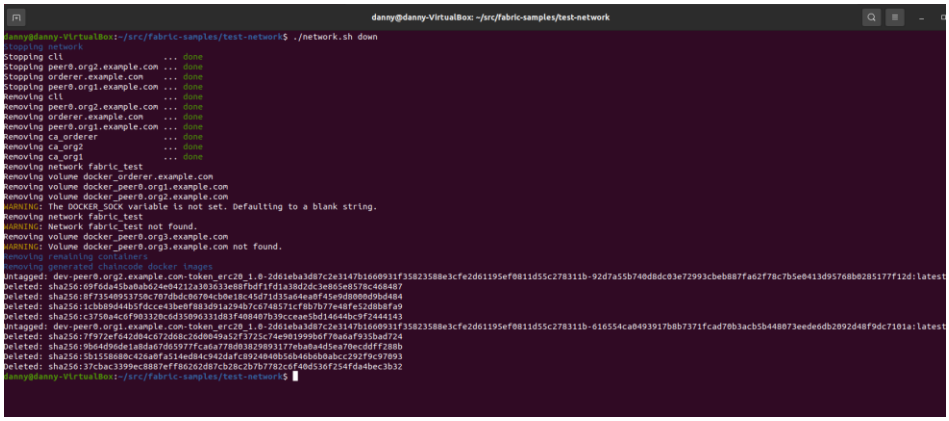
    inventoryWithRemovedAsset <- submit alice do
      exerciseCmd inventoryWithAsset RemoveAsset with asset = adjustedAsset

    shouldHaveRemoved <- submit alice do
      exerciseCmd inventoryWithRemovedAsset CheckIfAdded with asset = adjusted
Asset

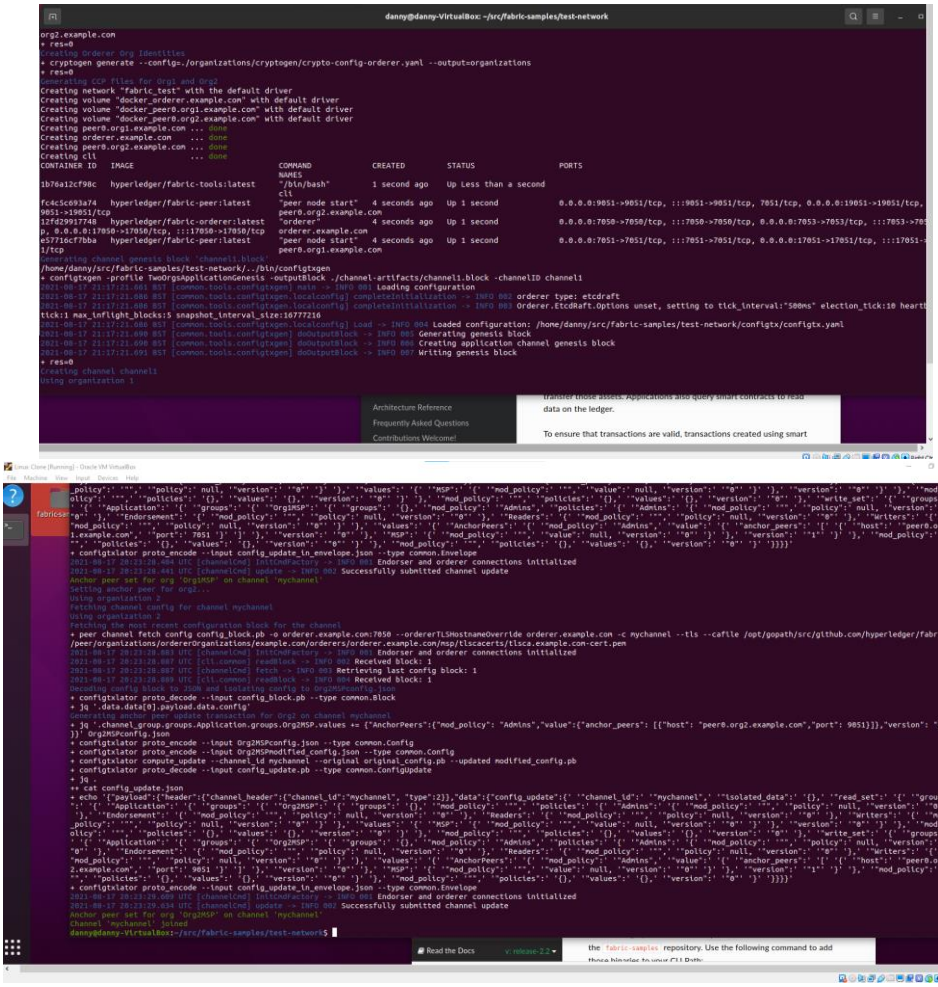
    shouldHaveRemoved === False
    return ()
```

Appendix 3 : Smart contract Testing

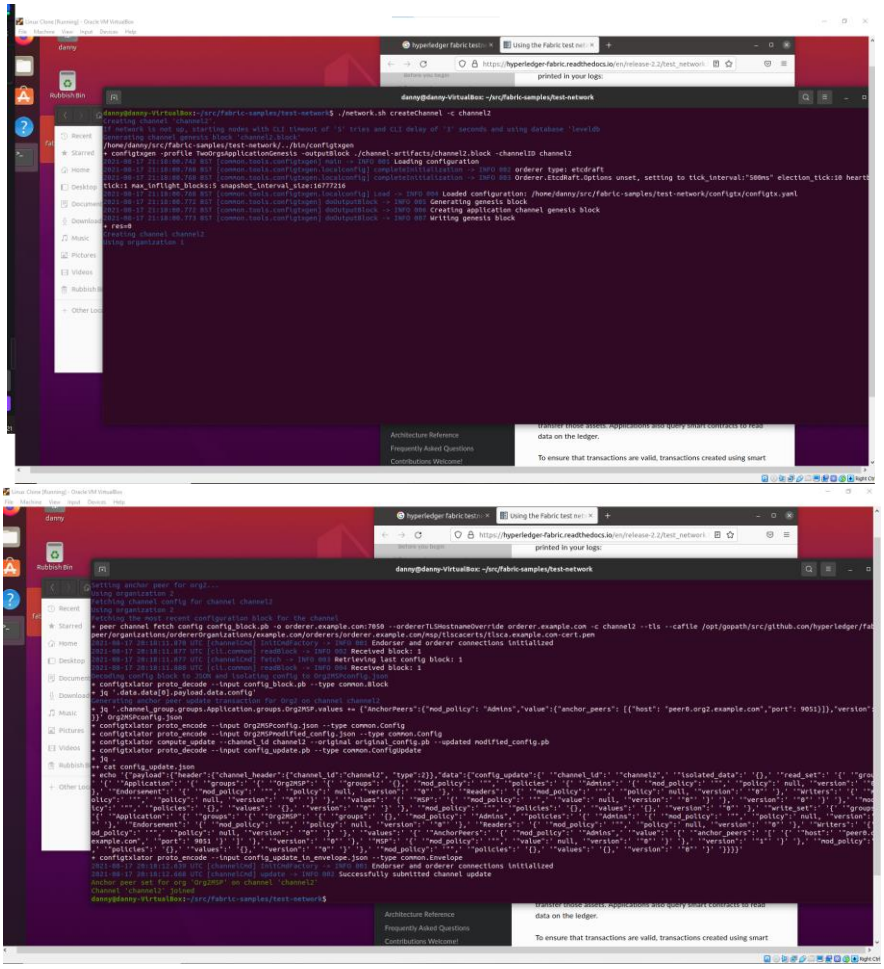
3A: Hyperledger Fabric Go Tests



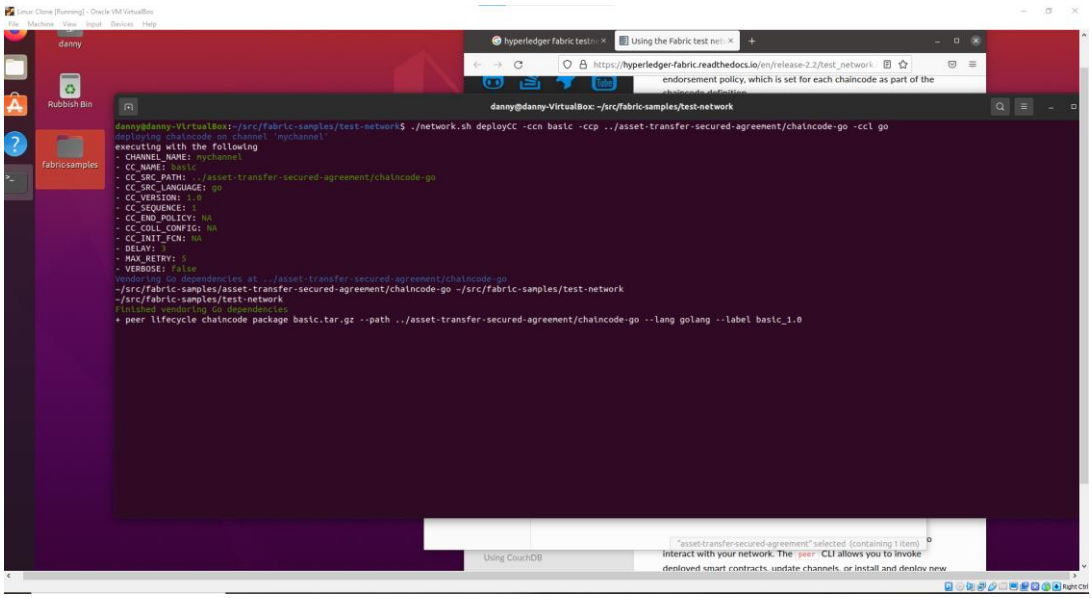
Shutting down test-network , removes dockers running



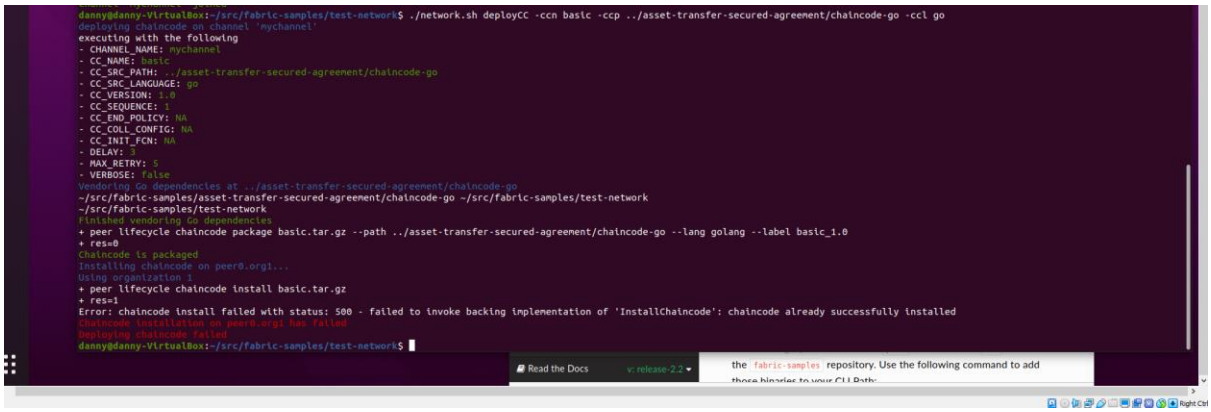
Channel 1 joined network



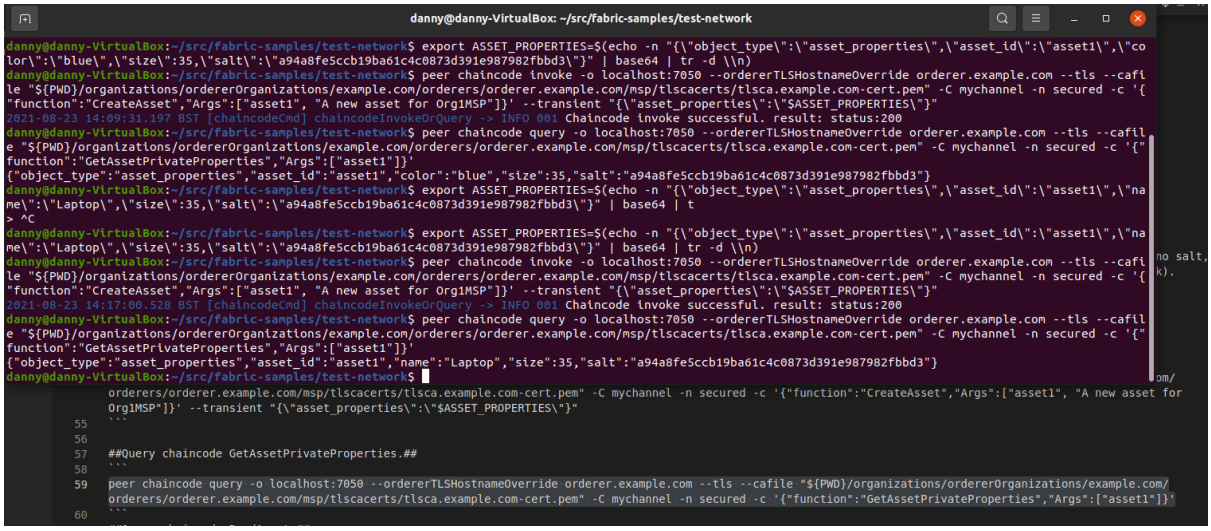
Channel 2 Joined network



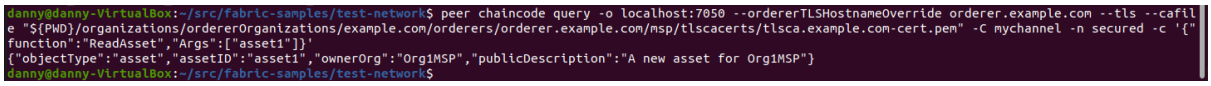
Deploying asset transfer chaincode to test-network



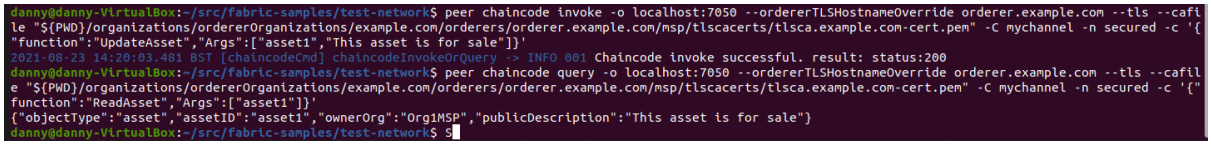
Redeploying chaincode again will prompt errors



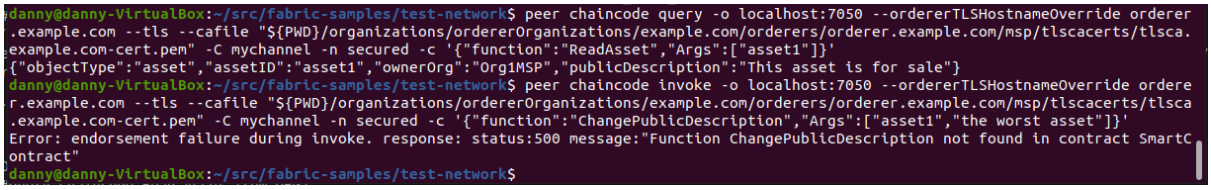
Invoke and query Chaincode Create Asset (Private details ORG1)



Read Public Asset



Setting Asset for sale



Org2 Reading asset from org1, attempting to modify asset (Should not be allowed)

```
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride ordere
r.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca
.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ChangePublicDescription","Args":["asset1","the worst asset"]}'
Error: endorsement failure during invoke. response: status:500 message:"Function ChangePublicDescription not found in contract SmartC
ontract"
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ export ASSET_PRICE=$(echo -n "{\"asset_id\":\"asset1\",\"trade_id\":\"109f4
b3c50d7b0df729d299bc6f8e9ef9066971f\",\"price\":110}" | base64 | tr -d \\n)
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride ordere
r.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca
.example.com-cert.pem" -C mychannel -n secured -c '{"function":"AgreeToSell","Args":["asset1"]}' --transient "{\"asset_price\":\"$ASS
ET_PRICE\"}"
Error: endorsement failure during invoke. response: status:500 message:"a client from Org2MSP cannot sell an asset owned by Org1MSP"
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ s
```

Org2 attempt to sell org1s asset showing security is in place

```
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafi
le "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{
"function":"AgreeToSell","Args":["asset1"]}' --transient "{\"asset_price\":\"$ASSET_PRICE\"}"
2021-08-23 14:27:10.928 BST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafil
e "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"
function":"GetAssetSalesPrice","Args":["asset1"]}'
{"asset_id":"asset1","trade_id":"109f4b3c50d7b0df729d299bc6f8e9ef9066971f","price":200}
danny@danny-VirtualBox:~/src/fabric-samples/test-network$
```

Org1 set asset sale price (200)

```
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafil
e "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"
function":"SetInspection","Args":["asset1"]}' --transient "{\"asset_properties\":\"$ASSET_PROPERTIES\"}"
true
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ export ASSET_PRICE=$(echo -n "{\"asset_id\":\"asset1\",\"trade_id\":\"109f4b3c50d7b0df729d299bc6f8e9e
f9066971f\",\"price\":100}" | base64 | tr -d \\n)
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafi
le "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{
"function":"AgreeToBuy","Args":["asset1"]}' --transient "{\"asset_price\":\"$ASSET_PRICE\"}"
2021-08-23 14:29:37.166 BST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafil
e "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"
function":"GetAssetBidPrice","Args":["asset1"]}'
{"asset_id":"asset1","trade_id":"109f4b3c50d7b0df729d299bc6f8e9ef9066971f","price":100}
danny@danny-VirtualBox:~/src/fabric-samples/test-network$
```

ORG2 Asset bid price (100)

```
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafil
e "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"
function":"GetAssetSalesPrice","Args":["asset1"]}'
{"asset_id":"asset1","trade_id":"109f4b3c50d7b0df729d299bc6f8e9ef9066971f","price":200}
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafil
e "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"
function":"GetAssetBidPrice","Args":["asset1"]}'
{"asset_id":"asset1","trade_id":"109f4b3c50d7b0df729d299bc6f8e9ef9066971f","price":100}
```

Query Asset sales and bid price

```
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganiz
ations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"AgreeToSell","Args":["asset1"]}' --transient "{\"asset_price\":\"$
ASSET_PRICE\"}"
2021-08-23 14:27:10.928 BST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganiz
ations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"TransferAsset","Args":["asset1","Org2MSP"]}' --transient "{\"asset
_properties\":\"$ASSET_PROPERTIES\",\"asset_price\":\"$ASSET_PRICE\"}" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.exa
mple.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt"
2021-08-23 14:43:19.609 BST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

Agree to bid offer and transfer asset to org2

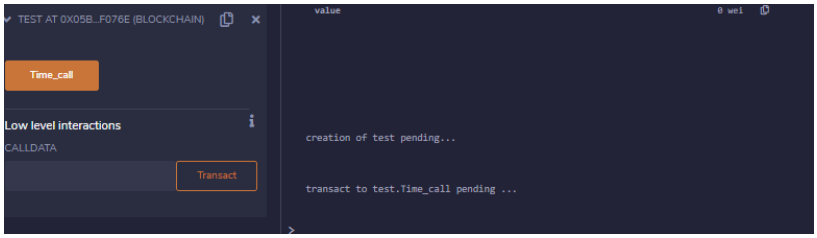
```
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer
.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.
example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadAsset","Args":["asset1"]}'
{"objectType":"asset","assetID":"asset1","ownerOrg":"Org2MSP","publicDescription":"This asset is for sale"}
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer
.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.
example.com-cert.pem" -C mychannel -n secured -c '{"function":"GetAssetPrivateProperties","Args":["asset1"]}'
{"object_type":"asset_properties","asset_id":"asset1","name":"Laptop","size":35,"salt":"a94a8fe5ccb19ba61c4c0873d391e987982fbbd3"}
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride ordere
r.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca
.example.com-cert.pem" -C mychannel -n secured -c '{"function":"UpdateAsset","Args":["asset1","This asset is not for sale"]}'
2021-08-23 14:44:56.413 BST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer
.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.
example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadAsset","Args":["asset1"]}'
{"objectType":"asset","assetID":"asset1","ownerOrg":"Org2MSP","publicDescription":"This asset is not for sale"}
danny@danny-VirtualBox:~/src/fabric-samples/test-network$
```

ORG2 is the new owner

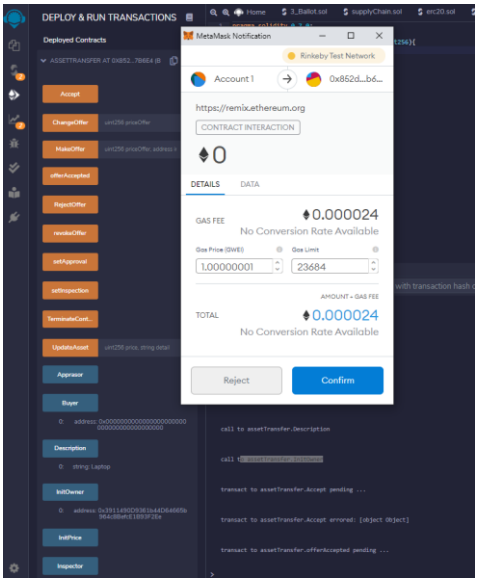
```
danny@danny-VirtualBox:~/src/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafi
le "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{
"function":"AgreeToSell","Args":["asset1"]}' --transient "{\"asset_price\":\"$ASSET_PRICE\"}"
Error: endorsement failure during invoke. response: status:500 message:"a client from Org1MSP cannot sell an asset owned by Org2MSP"
danny@danny-VirtualBox:~/src/fabric-samples/test-network$
```

Verify that org1 cannot attempt to sell asset Org2 owns

3B: Solidity Tests

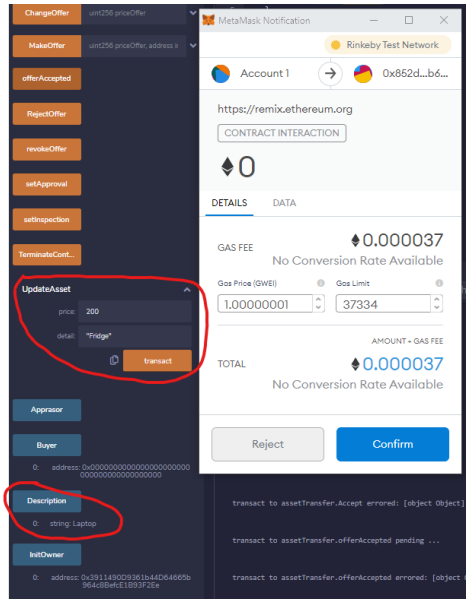


Pending transaction once function called on Ethereum testnet



Write function costs asset transfer

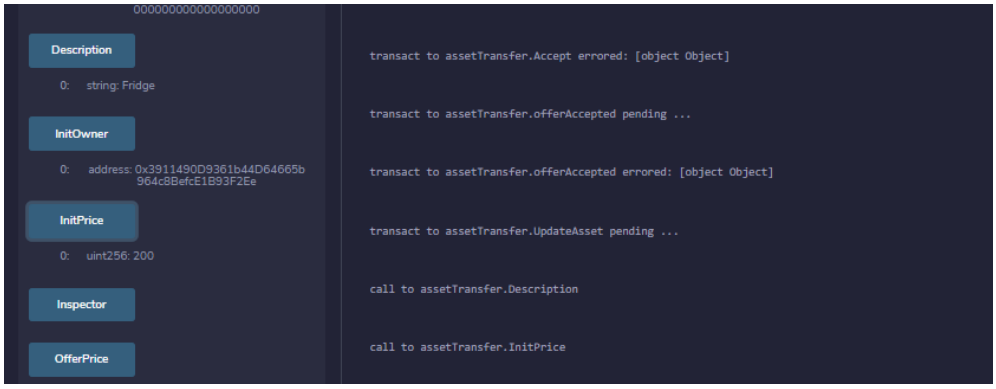
The cost varies depending on the type of function to be called, the read functions don't cost anything and they are near instant whereas modifying or calling a function which alters the contract will require a cost.



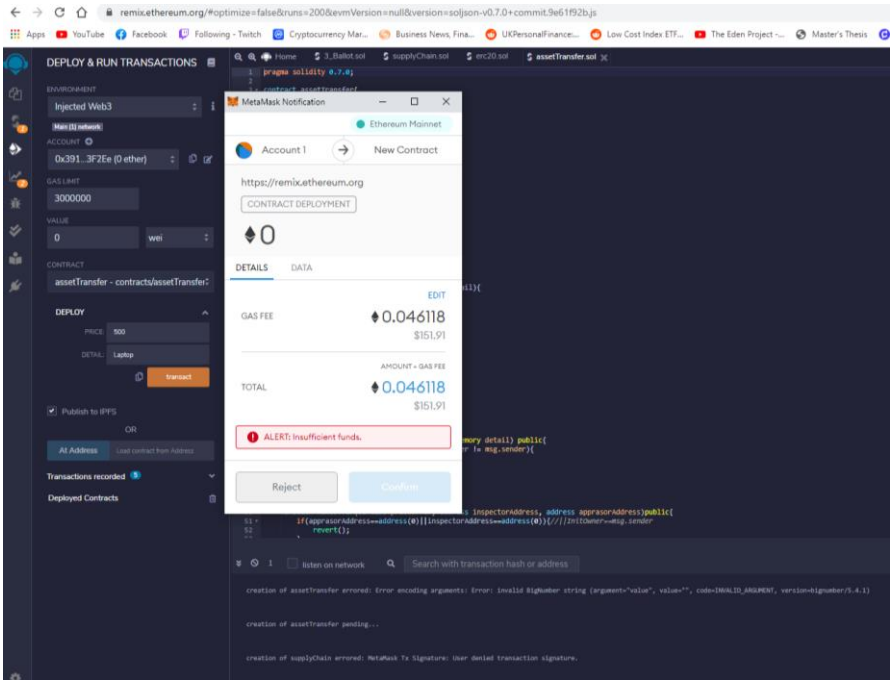
When updating a contract it requires a cost

When the transaction is complete it takes approximately 15 seconds on the Rinkeby test net to update the contact. On the ethereum mainnet this would take longer as the mainnet is designed to be more secure with its consensus.

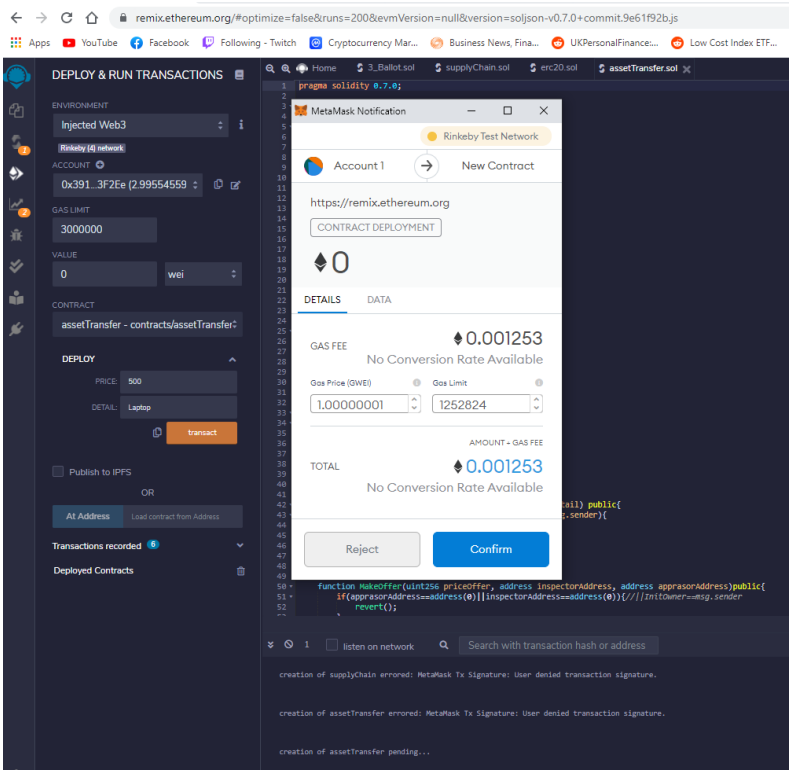
After the contract is updated we can call the read function and the string should be updated from “Laptop” to “Fridge”.



Ethereum Transaction Updated description and cost on Rinkeby network.



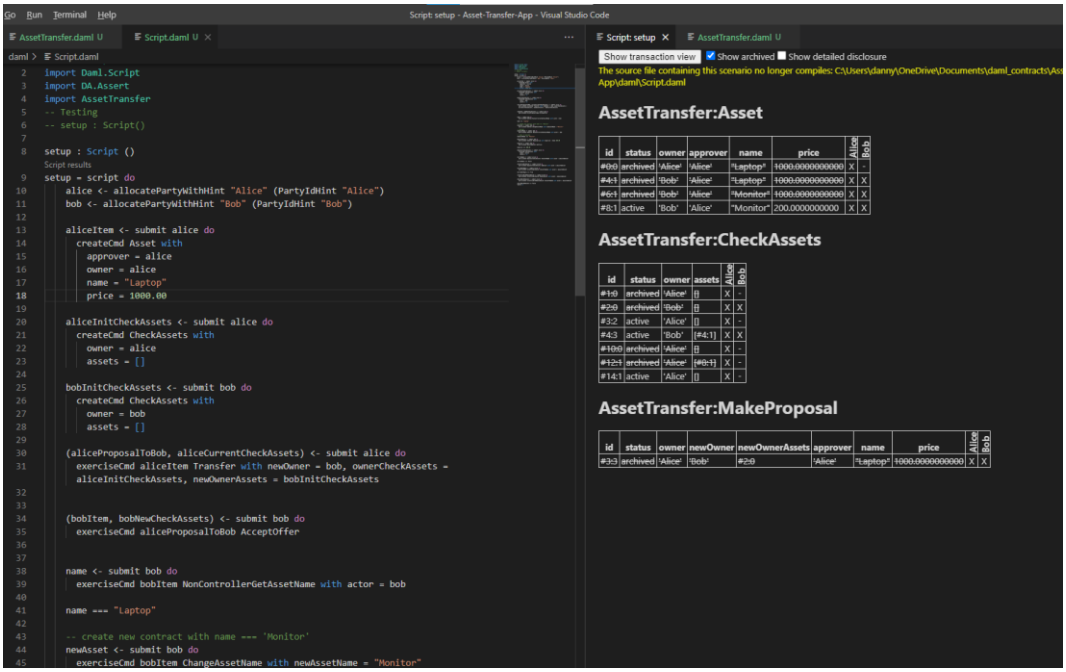
Asset Transfer Mainnet cost



Asset Transfer Cost on Rinkeby Testnet

3C: Daml Tests

Shown in the script file in appendix 2C



DAML Asset transfer success

3D: README file Hyperledger Fabric Commands:

[Secured asset transfer in Fabric Tutorial]

(https://hyperledger-fabric.readthedocs.io/en/latest/secured_asset_transfer/secured_private_asset_transfer_tutorial.html)

#Change to directory to run network#

```
cd fabric-samples/test-network
```

#Shut down network#

Network will shutdown and remove all data stored on ledger

```
./network.sh down
```

#Build/Run hyperledger network create channel called mychannel for chaincode#

```
./network.sh up createChannel -c mychannel
```

#Deploy Smart Contract to mychannel#

```
./network.sh deployCC -ccn secured -ccp ../asset-transfer-secured-agreement/chaincode-go/ -ccl go -ccep "OR('Org1MSP.peer','Org2MSP.peer')"
```

#Setting Environment Variables#

##ORG1##

```
export PATH=${PWD}/../bin:${PWD}:$PATH

export FABRIC_CFG_PATH=$PWD/../config/

export CORE_PEER_TLS_ENABLED=true

export CORE_PEER_LOCALMSPID="Org1MSP"

export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
```

export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE_PEER_ADDRESS=localhost:7051

```

##ORG2##

```

export PATH=${PWD}/../bin:${PWD}:$PATH

export FABRIC_CFG_PATH=$PWD/../config/

export CORE_PEER_TLS_ENABLED=true

export CORE_PEER_LOCALMSPID="Org2MSP"

export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

export CORE_PEER_ADDRESS=localhost:9051


```

#Operate org1 from terminal#

The "salt" parameter is a random string that would prevent another member of the channel from guessing the asset using the hash on the ledger. If there was no salt, a user could theoretically guess asset parameters until the hash of the of the guess and the hash on the ledger matched (this is known as a dictionary attack).

```

export ASSET_PROPERTIES=$(echo -n "{\"object_type\":\"asset_properties\",\"asset_id\":\"asset1\",\"color\":\"blue\",\"size\":35,\"salt\":\"a94a8fe5ccb19ba61c4c0873d391e987982fbbd3\"}" | base64 | tr -d \\n)


```

##Invoke chaincode call function create asset.##

```

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"CreateAsset","Args":["asset1", "A new asset for Org1MSP"]}' --transient "{\"asset_properties\":\"$ASSET_PROPERTIES\"}"
```

##Query chaincode GetAssetPrivateProperties.##

```
peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"GetAssetPrivateProperties","Args":["asset1"]}'
```

##Query chaincode ReadAsset.##

```
peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadAsset","Args":["asset1"]}'
```

##Invoke chaincode change description of asset e.g "asset for sale" function UpdateAsset##

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"UpdateAsset","Args":["asset1","This asset is for sale"]}'
```

##Query ReadAsset##

```
peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tls
```

cacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadAsset","Args":["asset1"]}'

```

#ORG2#

##Query Chaincode Read Asset from ORG2

```

peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadAsset","Args":["asset1"]}'

```

##Invoke and test ORG2 - Should NOT have access to modifying ORG1 Asset

```

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ChangePublicDescription","Args":["asset1","the worst asset"]}'

```

#Agree to sell asset set price at 110# (SHOULD ERROR)

```export ASSET_PRICE=$(echo -n "{\"asset_id\":\"asset1\",\"trade_id\":\"109f4b3c50d7b0df729d299bc6f8e9ef9066971f\",\"price\":110}" | base64 | tr -d \\n)

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"AgreeToSell","Args":["asset1"]}' --transient "{\"asset_price\":\"$ASSET_PRICE\"}"

```

##Query check asset price##

```peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"GetAssetSalesPrice","Args":["asset1"]}'

```

#Agree to buy as ORG2 OPERATE Terminal 2#

##Invoke buy offer##

```

export ASSET_PROPERTIES=$(echo -n "{\"object_type\":\"asset_properties\",\"asset_id\":\"asset1\",\"color\":\"blue\",\"size\":35,\"salt\":\"a94a8fe5ccb19ba61c4c0873d391e987982fbbd3\"}" | base64 | tr -d \\n)

peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"SetInspection","Args":["asset1"]}' --transient "{\"asset_properties\":\"$ASSET_PROPERTIES\"}"

Export ASSET_PRICE=$(echo -n "{\"asset_id\":\"asset1\",\"trade_id\":\"109f4b3c50d7b0df729d299bc6f8e9ef9066971f\",\"price\":100}" | base64 | tr -d \\n)

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"AgreeToBuy","Args":["asset1"]}' --transient "{\"asset_price\":\"$ASSET_PRICE\"}"

```

##Agreed purchase price ##

```

peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"GetAssetBidPrice","Args":["asset1"]}'

```

#Transfering asset from ORG1 to ORG1#

##Terminal 1 ORG1 Transfer Asset##

Operate from the Org1 terminal. The owner of the asset needs to initiate the transfer. Note that the command below uses the --peerAddresses flag to target the peers of both Org1 and Org2. Both organizations need to endorse the transfer.

```

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"TransferAsset","Args":["asset1","Org2MSP"]}' --transient "{\"asset_properties\":\"$ASSET_PROPERTIES\",\"asset_price\":\"$ASSET_PRICE\"}" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt"
```

##ORG1 drops price to 100 so both match

```
export ASSET_PRICE=$(echo -n "{\"asset_id\":\"asset1\",\"trade_id\":\"109f4b3c50d7b0df729d299bc6f8e9ef9066971f\",\"price\":100}" | base64 | tr -d \\n)

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"AgreeToSell","Args":["asset1"]}' --transient "{\"asset_price\":\"$ASSET_PRICE\"}"
```

##Call function again

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"TransferAsset","Args":["asset1","Org2MSP"]}' --transient "{\"asset_properties\":\"$ASSET_PROPERTIES\",\"asset_price\":\"$ASSET_PRICE\"}" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt"
```

##Check query

ORG2 should be asset owner

```
```

```
peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadAsset","Args":["asset1"]}'
```

#Update Asset Descirption for org2

```
peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"GetAssetPrivateProperties","Args":["asset1"]}'
```

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"UpdateAsset","Args":["asset1","This asset is not for sale"]}'
```

##Query for final check

```
peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadAsset","Args":["asset1"]}'
```