

FSCBlock: Designing Financial Smart Contracts on Permissioned and Public Blockchains

by

Muskan Vinayak

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
January 2019

© Copyright 2019 by Muskan Vinayak

Thesis advisor

Author

Dr. Ruppia K. Thulasiram

Co-Advisor: Dr. Srimantoorao S. Appadoo

Muskan Vinayak

FSCBlock: Designing Financial Smart Contracts on Permissioned and Public Blockchains

Abstract

Blockchain technology, though still at its infancy, is disrupting current business models by making intermediary services obsolete and the term has become a buzzword worldwide. A lot of research is going on to harness its full potential in many fronts from small to large corporate businesses. Currently, the Collateral Contract Services (CCS) are manually processed in financial institutions. The main objective of this research is to choose the most appropriate financial instrument, specifically derivatives, for CCS and allow its automated trading using Blockchain technology, which we have achieved on two prominent Blockchain platforms. In the first part of the thesis, formulating one of the derivatives, options, as a smart contract has been studied, wherein I have successfully generated and analyzed an options smart contract for Ethereum (public). Then, in the second part of the thesis, designing a Chain-code for CCS is researched using Hyperledger Fabric (permissioned), maintaining a transparent distributed ledger for the CCS between financial institutions.

Contents

Abstract	ii
Table of Contents	iv
List of Figures	v
List of Tables	vi
Acknowledgments	vii
Dedication	ix
1 Introduction	4
1.1 Financial Option	6
1.2 Collateral Services	9
1.3 Thesis Organization	10
2 Blockchain	11
2.1 Categories of Blockchain	11
2.2 Applications of Blockchain	12
2.3 Recent Use Cases for Blockchain in Financial World	15
2.4 Consensus Protocols	16
2.4.1 Proof-of-Work (PoW)	16
2.4.2 Proof-of-Stake (PoS)	17
2.4.3 Proof-of-Activity (PoA)	18
2.4.4 Ripple Consensus Protocol	19
2.5 Common Blockchain Platforms	20
2.5.1 Ethereum	20
Cost Effectiveness	23
Managing Techniques	23
Prediction Market	24
Oracles	24
Possible Vulnerabilities	25
2.5.2 Hyperledger	28
3 Related Work and Problem Statement	34

4	Solution Methodology	38
4.1	Methodology adopted for Options Contract on Ethereum	38
4.1.1	Creation of Smart Contracts	38
4.1.2	Analyzing smart contracts	44
4.2	Methodology adopted for implementation on Hyperledger	44
5	Experiments and Results	50
5.1	Design outcomes of Smart Contract on Ethereum	51
5.2	Design Outcomes of Smart Contract on Hyperledger blockchain	58
6	Conclusion and Future Work	64
	Bibliography	68

List of Figures

4.1	Collateral Blockchain Network	46
4.2	Collateral Blockchain Network Components	48
4.3	Large Scale Collateral Blockchain Network	49
5.1	Contract Creation	52
5.2	BuyCallOption - error	53
5.3	BuyCallOption	53
5.4	Successful Transaction for Buy Call Option	54
5.5	Successful Transaction for Write Call Option	55
5.6	Steps for deployment on Ethereum Ropsten Testnet	56
5.7	Collateral Network Setup	58
5.8	Successful Admin enrollment and user registration for FinBank	59
5.9	Successful Admin enrollment for TechFin	59
5.10	Successful Admin enrollment for FinIns	59
5.11	Successful User registration for TechFin	59
5.12	Successful User registration for FinIns	60
5.13	Collateral Creation by Alice	60
5.14	Collateral Query Result by Alice	60
5.15	Collateral Used to pay margin	61
5.16	Collateral Query Result by Alice after paying collateral	61
5.17	Collateral Query Result by Manet	61
5.18	Collateral Creation Result by Alice on Channel 2	62
5.19	Collateral Query Result by John	62
5.20	Error produced in Query Result by John	62

List of Tables

5.1 Vulnerabilities identified using Oyente	57
---	----

Acknowledgments

On this moment of my thesis submission there are not enough words in my dictionary to express my gratitude to all, whom I have come across in this path of my journey and learned to live the life. My sincere utmost thanks to my advisor Dr. Ruppa K. Thulasiram and co-advisor Dr. Srimantoorao S. Appadoo for their kind support throughout my Masters journey with a positive feedback. This research would not have been possible without their immense guidance, extensive encouragement and confidence towards me. I am indebted to you for whatever you have done for me.

Beside my advisors, I would like to thank my committee members Dr. Pourang Irani and Dr. Saman Muthukumarana for their insightful comments and technical and constructive feedback. I would also like to acknowledge Dr. Parimala Thulasiraman who always inspired me with her positive encouragement.

I would like to acknowledge the timeless efforts, sacrifices, and motivation provided by my grandfather Rajinder Paul Vinayak, my grandmother Swaran Vinayak, my father Varinder Pal Vinayak and my late mother Asha Vinayak, who are the pillars of my strength. A special thanks to my uncle Rahul Kumar and my aunt Dr. Anju Bajaj for always providing me with right guidance in a friendly manner and never letting me down. Most importantly thanks to my brother Vardan Vinayak and cousins Kartik Vinayak and Swastik Vinayak for always cheering me up. My family members supported me in all possible ways while completion of this work and believe in my dreams.

I would like to thank my lab-mates, Anurag, Khatereh, Manmohit, Saulo, Wayne, Ziyue, and fellow colleagues, for always being there in both happy and crisis moments.

I was lucky to be a part of such a group filled with learning moments. Also, my special thanks to my friends, especially Payal and Shivam, from back home for always encouraging and believing in me.

Last but not the least, I would like to thank Almighty God for giving me opportunities to explore my knowledge and guiding my path.

*This thesis is dedicated to My Grandmother and Grandfather for their
immense love and support.*

Glossary

Solidity Solidity is a high-level language that is used to create smart contracts for Ethereum.

Ether This is the cryptocurrency used in the Ethereum Blockchain

Wei The smallest unit of Ethereum's currency Ether. $1 \text{ Ether} = 10^{18} \text{ Wei}$

Turing-complete A system is said to be Turing-complete if it can perform any possible computation demanding undefined amount of memory.

gas gas is the amount paid to carry out any computation on Ethereum Blockchain. This can be purchased using Ether

nonce A random number that is used in generating the block hash that satisfies the target number of leading zeroes.

Primecoin Primecoin is a cryptocurrency in which the Proof-of-Work involves finding a prime number chain.

PPcoin This is another cryptocurrency system which utilizes both Proof-of-Work and Proof-of-Stake.

BlackCoin Another cryptocurrency whose consensus mechanism relies on Proof-of-Stake.

UTXO These represent the coins that can be spent by the owner.

Ethereum Virtual Machine (EVM) EVM is an environment in Ethereum where the smart contract are executed.

pricegeth This is used to fetch exchange price data from exchange API and provide it to the smart contracts when requested.

Oraclize Oraclize is one of the oracles that provides data to different Blockchain platforms including Ethereum.

oracles An oracle is a service through which the external world data can be fetched into the contract.

F* This is a functional programming language.

Hyperledger Fabric Hyperledger Fabric is a Blockchain platform and a part of the Hyperledger projects.

chaincode A chaincode is a program in Hyperledger Fabric that defines the business rules agreed by the parties involved in the network.

Kafta or solo These are the consensus protocols that can be used in the ordering service of Hyperledger Fabric.

BTCETH This is the exchange rate between Bitcoin and Ethereum currency.

mist This is a browser for Ethereum.

geth It is command line tool for executing and interacting with a full Ethereum node.

Oyente It is a tool that can be used to find vulnerabilities in Ethereum smart contracts.

golang Go is a programming language.

Chapter 1

Introduction

Blockchain is a decentralized ledger consisting of a series of blocks that contain different transactions. This ledger can only be appended with new blocks and acts as a source of trust between unknown parties. The concept was initiated in early 2009 by Satoshi Nakamoto [1] based on some prior work done by Vishnumurthy et al. [2].

Blockchain is maintained and regulated by a network of miners. Each of these miners perform the protocol designed for achieving consensus and appending a block to the chain. For instance, in the Bitcoin blockchain, the miners conduct proof-of-work to verify the authenticity of a new block to be added to the blockchain. After completing the proof-of-work, a miner sends it to the network to obtain a consensus before adding it to the blockchain [1]. Although the concept of blockchain was initially intended for cryptocurrencies like Bitcoin, this technology is now being used in other areas for instance, to create decentralized applications through smart contracts [3]. The idea of having a decentralized digital form of currency is being studied for decades but was not successful until recently as the researchers were unable to remove entire

element of centralization in the past [4]. Some of these works include the 1980s and 1990s anonymous e-cash protocols and the b-money concept by Wei dai [4]. With the increasing popularity of Bitcoin, different projects were developed based on Bitcoin's underlying technology. These include Namecoin project [5], alt-coins [6], etc. Another major impact of blockchain technology is on smart contracts that can be used to create user-defined decentralized applications. Smart contracts are computer algorithms that define the rules of the execution of the service provided through these contracts. These can be used to create different applications such as decentralized voting, crowd-funding, financial swaps, to name a few. The idea of smart contract is not new and was proposed by Nick Szabo [7] in 1997, before even the introduction of the Bitcoin. Smart contract comprises of an algorithm, current state of the contract and history of states. These smart contracts exist on Ethereum, Hyperledger and other blockchain platforms.

Ethereum [8] is a project through which decentralized “transaction-based state machines” can be developed. Ethereum blockchain is more generalized than Bitcoin and different business rules can be stored on it as Turing-complete algorithms. These algorithms are stored on Ethereum in the form of *EVM* (Ethereum Virtual Machine) bytecode [9]. The developer can write these smart contracts using languages such as *Solidity*, however, these are converted into EVM bytecode before being deployed on the Ethereum blockchain. The functionalities of these smart contracts can be utilized either internally by other contracts from within the blockchain or externally through transactions. The intrinsic currency of Ethereum is known as *Ether*, which further has several sub-denominations, the smallest being *Wei* [8]. To execute any

computation on Ethereum, a fee has to be paid in terms of Ethereum's payment mode i.e. *gas*, purchased using *Ether* [8]. This avoids the possibility of abusing the network [8]. Smart contracts can be explored to convert some real world applications into decentralized applications such as option contract, credit default swap contract, voting system [10] [11] and independent applications for governance (for instance, outsourced computation [12], decentralized gambling [13]).

Ethereum is a public blockchain. That is, anyone can start a miner node and maintain the blockchain. The main aim of designing applications on Ethereum is to remove the dependency on centralized parties. For instance, in financial derivative markets, there exists a centralized party for clearing and settlement of contracts between investing parties. The focus of my research is two-fold: In the first part, I study Options, one of the financial derivatives, using smart contracts on the Ethereum blockchain. Then, I study another financial application contract, designed for collateral services, using another blockchain platform known as Hyperledger.

1.1 Financial Option

A financial option is an agreement between two investing parties, where one party (holder) gets the rights to exercise the option at a specific time (expiration) and the other party (writer) is obliged to the decision of the first party. Exercising an option means buying or selling an underlying asset such as stock. There are two types of options namely, put and call options [14]. In a call option, the holder gets the right to buy the underlying asset at a specific price during a certain time, whereas in a put option, the holder gets the right to sell the underlying asset during a certain time

period and for a specific price. This specific price is known as strike price. There are different styles of options depending upon how they are exercised. The option can either be exercised anytime before or on the expiration date (American style) or only at the expiration time (European style) [14]. These two are simple styles of options. There exists many other complex styles through which an option can be exercised. To own these rights, the buyer (holder) of the option has to pay a premium amount to the seller (writer).

Clearing corporations act as central authority in monitoring these contracts to make sure that the obligations are honored according to the contracts. However, using the decentralized technology such as blockchain, raises a risk of defaults associated with any application where money is involved. Hence, to utilize the blockchain technology we have adapted an approach to handle financial options that eliminate the possibility of defaults and making it possible to execute it on the Ethereum blockchain without the need of centralized clearing agency.

Computing the option value to determine if the premium to be paid for the option contract is worth or not is not the focus of my research. This can be done outside the blockchain before deciding to enter into the contract. As computation on Ethereum costs *gas* to be bought using *Ether*, repeating the same computation on all nodes in the Ethereum network to compute option price is expensive in terms of time and money. Hence, I am not considering pricing of options in my thesis. Also, the main purpose of blockchain is to allow decentralization and the option price can be computed by the parties using various financial models and doesn't require any decentralization.

The smart contracts on Ethereum blockchain are vulnerable to attacks by ma-

licious users. One of the reasons that motivates adversaries to attack the smart contracts is that the smart contracts hold some amount of money that can be gained through attacks. There has been certain incidents in the past highlighting the malfunctioning of smart contracts. These include the unexpected execution of smart contracts or thousands of dollars value of virtual coins being locked away [15]. Atzei et al. [16] have presented a detailed survey of such possible vulnerabilities. These vulnerabilities might lead to a huge loss as seen in the Decentralized Autonomous Organization (DAO) bug, which caused loss of few million dollars [15]. In addition to the vulnerabilities, there are other risk factors that need to be studied before entering the cryptocurrency market. An incident occurred recently where the death of Gerald Cotten, the CEO of QuadrigaCX, resulted in \$145 million of bitcoins and some other digital assets being locked away [17]. The company stated that Cotten was the only person having access to these assets [17]. This demands for exploring possible vulnerabilities in a contract before it's actual deployment in order to reduce the risk of attacks. In the first part of my research, I use an open source tool called Oyente [15] to study possible vulnerabilities in the adapted (options) smart contract that I create. Oyente [15] works with the EVM (Ethereum Virtual Machine) bytecode rather than the high-level language used to write contracts.

In Ethereum anyone can be a miner and the node held by that miner contains a copy of blockchain and can view its data. However, the enterprises might be reluctant to put their data on a public blockchain. This data may include their financial asset holdings or the price charged for different services being offered to clients, to name a couple. Hence, the industries are exploring the other kind of blockchain known

as permissioned blockchain. In this kind of blockchain, the identity of the nodes maintaining blockchain is known. Also, the participants in the blockchain network and the operations they can perform on the data is controlled.

The two major founding principles in the first part of my research are option contracts and blockchain technology. In the first part of my thesis, I am studying the process of exchanging profit between parties involved in an option contract on Ethereum blockchain.

1.2 Collateral Services

The main teams in a typical financial organization include Trading, Management, Accounting/Reporting and Senior management with roles as follows:

- The trading department's responsibility is to handle trades for the organization and deal with other financial houses and exchanges. The nature of the trade depends on two factors: the priorities of the organization and the optimization of their investment portfolio.
- The Accounting/Reporting department is accountable for maintaining the books of all the trades and preparing reports for board members and the shareholders.
- The Senior management team's task is to administer the overall functioning of the business.

One of the activities handled by the financial business organizations is the Collateral Contract Services (CCS). This means exchanging collateral with other organizations to cover the day-to-day margin commitments. The collateral could be of

different forms such as cash, equities, derivatives, to name a few. The CCS are handled manually in financial institutions these days and making them smart demands handling security issues. The two major founding principles in the second part of my research are financial collateral contracts and blockchain technology. The key objectives of my research are (1) use a pool of multiple securities comprising of financial instruments, especially, derivatives; (2) formulate different sample CCS each with different specifications; (3) algorithm design in identifying and selecting most appropriate instrument for the CCS and trade it in an automated manner; and (4) appropriate use of blockchain technology in achieving these objectives.

For the second part of my thesis, I study how the blockchain can be utilized in achieving transparency and automation in CCS. The day-to-day collateral exchange activities are recorded on the permissioned blockchain named Hyperledger.

1.3 Thesis Organization

The thesis is organized in the following manner: In Chapter 2, I explain the general features of the blockchain, its types, use cases, two blockchain platforms studied in my thesis and some of the consensus protocols that are being studied in the literature. In Section 2.5.1 and 2.5.2, I provide details about the two different blockchains I used in my research, Ethereum and Hyperledger respectively. In Chapter 3, I discuss some of the closely related works and state my research problem. In Chapter 4, I explain the methodology I propose for this research. I describe my experimental setup and discuss results obtained from different experiments in Chapter 5. Finally, I conclude my study in Chapter 6 and highlight some of the future research directions.

Chapter 2

Blockchain

The concept of maintaining an immutable ledger of records has attracted various researchers and businesses. Although the concept was initially meant to be used with cryptocurrencies, it is constantly being explored for use in different areas to ease the business processes by acting as a source of trust.

2.1 Categories of Blockchain

A blockchain can be categorized into permissionless (public) and permissioned (private) blockchain based on the level of openness in the participation.

Permissionless Blockchain: In a permissionless blockchain, anyone can join the network and perform the mining process. The miners (network participants) compete with each other to mine the next block based on the protocols of the network.

Permissioned Blockchain: As the name suggests, permissioned blockchain is a kind of private blockchain where only the authorized people are allowed to join the

network. In other words, only the authorized people can verify the transactions and update the ledger.

2.2 Applications of Blockchain

A lot of research is being done studying the applicability of blockchain technology in different sectors such as banking, supply chain, finance, to name a few. Some of the real world applications include the following:

- IBM food trust [18] is a network built on the Hyperledger project that allows collaborations between retailers, distributors, processors, and growers. This solution [18] gives permissioned users an access to the data related to food supply chain. The entire process of the supply chain can be tracked using this data for quick analysis. This was an extension of the pilot project run for more than a year with major participants including Walmart, Nestle, Golden State Foods, etc. The project ensures trust in the food supply chain and helps in protecting consumers by tracking the issues. Walmart is planning on using this to track different products.
- Blockchain is also finding it's way in adding transparency and cost-efficiency in the airline processing systems. NIIT Technologies, designed a "Chain-m" blockchain application [18] for enhancing the passenger ticket processing. It has an interactive web interface and dashboard for users. For this purpose, hyperledger fabric is being utilized that stores a number of details like the number of tickets sold, commissions, fare charged, amount of taxes collected,

etc.

- Another use of hyperledger technology in supply chain is done by Cambio Coffee, which is an organic coffee seller. In their project [18], Hyperledger Sawtooth Blockchain was utilized. Through the use of hyperledger Cambio Coffee allowed its users to track the entire history of the coffee they are buying by scanning QR codes. The QR codes were added to the packaging of coffee from ScanTrust, which acted as an identity to update its movement through different steps ranging from harvesting to shipping.
- A non-profit Insurance organization designed an automated Blockchain solution [18] (“open Insurance Data Link”) to add efficiency and accuracy in the insurance regulatory reporting. In this reporting process, the insurance companies are supposed to securely report to the regulators some amount of regulatory data. This data must follow a number of compliance requirements. As mentioned in the article [18], this is the first project to gather and share statistical data. The insurers can put the data in compliance with the regulatory requirements on the ledger and the regulators can be provided access to the required data.
- Blockchain is also being studied extensively in Healthcare. An initiative [19] has been signed by IBM Watson Health with U.S. Food and Drug Administration to design a Blockchain based solution to share information on health. Through this system [19], a person can securely share his/her Healthcare data obtained from different sources such as clinical trials, mobile devices, IoT (Internet of

Things) on the Blockchain. This could help the Healthcare researchers and providers to provide an exclusive set of solutions as they can have a complete picture about the patients data. For the initial study, the focus of this project is only on the data related to oncology. In a recent paper [20], on the basis of a survey it has been noted that more than 70% healthcare industry innovators think that the healthcare system could benefit from Blockchain Technology in clinical records, regulatory compliance and medical or health records.

- Another Blockchain application named Yijian Blockchain Technology Application system [19] is designed on Hyperledger fabric by a Chinese conglomerate Hejia that helps to eradicate the financing challenges that the retailers in the pharmaceutical industry faces. This is achieved by having a transparent system for recording the drugs transactions involved during the supply chain. It involves Hejia and a few more participants such as hospital, retailer and bank.
- A company, named Medicinal Genomics, has proposed the idea of how the cannabis industries can utilize the blockchain technology and the concept of DNA sequencing to track the supply chain of weed for building trust [21]. By maintaining a genetic record of all the weeds could help in distinguishing the other illegal weed. The users can have a mobile application linked to blockchain through which they can track if the weed is legitimate using the QR code. This genetic data can be put by the cultivators and breeders to the blockchain which in the paper is Bitcoin blockchain.
- Ethereum blockchain can be used to create decentralized financial or Initial

Coin Offerings (ICOs) applications. Decentralized financial applications [22] use the Ethereum blockchain's smart contract concept for provisioning loans and permitting decentralized exchanges. "MakerDAO" [22], a project on Ethereum blockchain has created a stablecoin, which is backed by Ether. In the ICOs [22], the businesses can generate funds by selling tokens for ethers. The motivation of the individuals who buy it is that these tokens have use in the application. A ICO named Bancor collected \$153 million by selling tokens in around 3 hours.

2.3 Recent Use Cases for Blockchain in Financial World

- London Stock Exchange [23] has joined hands with IBM to create a blockchain solution that could facilitate the digital issuance of private securities. The distributed ledger will contain a record of all shareholder transactions. As mentioned in the article [23], this solution could help the small enterprises to connect and share information who otherwise face problems in linking to the public stock exchange. Another company named Barclays leveraged blockchain for testing the trading of derivatives [24]. In their project [24], R3 Corda Blockchain was utilized. Nasdaq has its own blockchain, called Nasdaq linq [25], that was used to issue private securities and proved to be time-efficient.
- Chile's largest stock exchange company was planning to use blockchain to automate the trading processes [26]. The regulators and banks will be maintaining the nodes in the blockchain network through which ownership information for

certain derivatives, such as stock, will be exchanged.

- Another use case of blockchain is by the TMX Group [27], the operator of many stock exchanges including Toronto Stock Exchange and Montreal Stock Exchange, where the blockchain is used to gather the opinions or votes of the shareholders. Also, a project [28] where blockchain is being used to track the entire transaction process of international trades is conducted by a company called UBS, a Swiss bank. This solution [28] enables to automate the issuance letter of credit in large transactions through programs that reduces the process time.

2.4 Consensus Protocols

As a network of miners is responsible for maintaining the blockchain, there exist protocols that are used to achieve consensus between them. These protocols decide how the miners can append the next block to the blockchain. Different blockchains follow different consensus protocols. Some of the protocols being used are explained below:

2.4.1 Proof-of-Work (PoW)

In this consensus protocol, the miners perform some computationally extensive task to append a block to the blockchain. In Bitcoin [1], the PoW resembles the Adam Back's Hashcash [29]. In Bitcoin [1], the miners look for a single time use value called *nonce*, such that the block's hash produced using that nonce (with SHA-

256) has specific number of leading bits as zeros. The amount of effort needed to compute this hash is exponential to the number of zero bits required. This difficulty is adjusted so that the average time to compute a block in Bitcoin is 10 minutes. Also, as the block contains the hash of the previous block, if a miner changes a block then that miner has to compute the hash of all the following blocks. When miners compute a block they get rewarded with some bitcoins [1].

Another project called Primecoin cryptocurrency [30] had a new concept of PoW mining. Instead of the usual calculation of Hashcash PoW, this protocol relies on finding prime numbers chain. These new prime number could be useful to scientists as well whereas the hashcash PoW is just useful for computation of block's hash. The prime chains have three categories each with specific characteristics. Also, the idea doesn't allow to reuse the PoW in next block [30].

2.4.2 Proof-of-Stake (PoS)

The main disadvantage of PoW is that it requires a lot of computational power to add a block to the blockchain. Another algorithm proposed in the literature is PoS in which the miners put their stake on the block instead. There have been different ways proposed on how PoS can be incorporated.

In the approach by King and Nadal [31], named *ppcoin* project, a hybrid design of using PoW and PoS is suggested. The authors [31] highlight that this will address the problem of a lot of energy consumption observed in PoW blockchain. The protocol relies on the *coin age* of the miner, which is calculated as the number of coins he/she holds times the number of days those coins have been held for. The initial minting

process in this approach [31], involves PoW, however, in a long run the significance of PoW lowers. A new field, called *timestamp*, is included in each transaction to leverage the *coin age* calculations. The hash target to achieve for adding a block is divided by the *coin age*. Hence, if a block has more coin age it takes less time to compute the hash. The *coin age* of the block is computed by summing up the *coin age* of all the transactions it comprises [31].

Another protocol, known as PoS 2.0 was proposed by Vasin [32] that is believed to resolve the possible issues of security in PoS blockchain. These issues include either carrying out a double-spend attack by the use of *coin age* or irregular use of stack by the honest nodes or the chances of regeneration of a future PoS [32]. This protocol is used in the *BlackCoin* cryptocurrency, which is purely based on PoS [32]. The major updates that this algorithm had for solving the above issues include removing the concept of *coin age* with just coins that demands more online availability of the miners, updating the stake modifier at every modifier interval, increase in the expected block time, and change in the block hash to SHA256d [32].

2.4.3 Proof-of-Activity (PoA)

In this protocol, the authors [33] proposed to make a blend of PoW with PoS algorithm. The protocol starts with the miners performing proof-of-work to produce an empty block header comprising of a *nonce*, miner's public address and the previous block's hash. After successful generation of the hash, the miner broadcasts this block in the network. This header determines the N stakeholders that will be required to validate this block using the "follow-the-satoshi" approach. After this, the stakehold-

ers who are part of these N stakeholders make sure if the header is valid and sign it using their private key of the currency that made them one of the stakeholders. When N^{th} stakeholder signs it, it extends the block header by adding as many transactions as possible. Finally, the block is broadcast to the network and the reward is distributed between the miners and stakeholders.

2.4.4 Ripple Consensus Protocol

David Schwartz et al. [34] proposed a low latency consensus protocol called Ripple based on the problem of Byzantine failures. The protocol achieves consensus in a distributed architecture using a set of trusted sub-network. There is a threshold on the malicious nodes that can be present in the sub-network. The authors explain how the three main categories of problems that are usually faced in a distributed payment system are addressed in Ripple. These categories are: (i) Correctness - ability of the protocol to identify correct and fraudulent transactions; (ii) Agreement - a method to achieve a global truth on a decentralized architecture; (iii) Utility - the applicability of the proposed payment method on real life situations regarding latency, computational power and energy required to the maintenance of the system itself. The Ripple protocol is composed of servers (responsible for participating in the consensus protocol), Ledger (keeps the balances of each user's account and keep them updated with transactions that achieved consensus), Last-closed ledger (the most recently updated ledger in the network), Open Ledger (maintains all transactions not submitted to the consensus protocol), Unique Node List (UNL is the sub-network responsible to achieve consensus and approve transactions), proposer (server that

are suggesting transactions to be validated by the consensus round in each UNL). The protocol comprises of different rounds. The time frame for nodes to propose transactions is 2 seconds. The transactions must receive a minimal consensus of 80% to be added to the closed ledger. To remove the possibility of having forks the algorithm used in Ripple protocol is network split detection. The Ripple protocol has been updated, and for the present state of the protocol readers can refer to the XPR ledger Consensus Protocol document [35].

2.5 Common Blockchain Platforms

2.5.1 Ethereum

Ethereum, one of the popular Blockchains, has a concept of Smart Contracts that has gained the attention of many researchers and businesses. Although the Bitcoin does support some smart contract concepts, however, these are just the minimal ones and related to cryptocurrency transactions [4]. Also, smart contract in Bitcoin do have some limitations such as not being Turing-complete, lack of possibility of maintaining states more than the two states (spent or unspent) and the UTXO do not know about the details of the data on the blockchain [4]. The Ethereum blockchain is a more generalized project than the Bitcoin blockchain, that was mainly used to serve as a ledger for cryptocurrency transactions. The primary aim of the Ethereum project [8] is to allow transactions between any two individuals who do not know each other and have no way to build trust between them. It stores Turing-complete algorithms in the form of EVM (Ethereum Virtual Machine) bytecode. Ethereum

can be seen as a state-machine, where the states are based on the transactions. In other words, Ethereum starts with an initial state called the genesis state and after the execution of different transactions reach a final state. The factor that evaluates the state in the Ethereum is accounts. There are two main types of accounts in the Ethereum namely, *externally owned accounts* (has a private key but no code) and *contract accounts* (consists of smart contract code) [4]. The state is changed only when an account sends a transaction to other accounts. The interactions between accounts in Ethereum occur through two modes: transactions and messages [4]. The externally owned accounts can send a transaction to any other accounts. On the other hand, contract accounts can send the messages only to other contract accounts. When a contract account receives any transaction or message it executes the code it holds.

As mentioned earlier, Ethereum's [8] intrinsic currency is known as *Ether* that has various sub-denominations with the smallest being *Wei*. The integer values of Ethereum's currency are calculated using *Wei* [8]. A user has to pay a fee in terms of Ethereum's payment mode i.e., *gas*, to perform any computations on Ethereum. This *gas* is purchased using the intrinsic currency of Ethereum and prevents the chances of abusing the network [8].

Currently, the miners in the Ethereum blockchain use the PoW blockchain for transaction validations and for appending a block. However, Ethereum blockchain is planned to move to Proof-of-Stake consensus protocol due to its advantages including more energy efficiency, centralization risks are decreased, and security to name a few [36]. This is through the Casper implementation. There are two designs for the

Casper Implementation: “Casper the Friendly Finality Gadget (FFG)” and “Casper the Friendly GHOST: Correct by Construction (CBC)”. Casper [37] is based on the Byzantine Fault Tolerant (BFT) style PoS, which mathematically proves that conflicting blocks are not finalized until and unless more than two-thirds of the network is honest. Casper offers several additional features to the BFT algorithm including punishing the validator (taking away validators deposit) who violates the protocol, allowing validator set to change dynamically and other defense mechanisms. The Casper protocol [37] will proceed in phases with the initial phase being a hybrid PoW/PoS system and then complete PoS. In this version [37], the validator set is fixed and protocol overlays the existing PoW proposal mechanism. The validators submit some deposit at the time of joining the network, which can increase or reduce as blocks are validated. The validators work on the subchain of the blocks based on checkpoints. This change has been deprecated as shown in the github repository [38]. Casper CBC [39] algorithm is based on the adapted GHOST (Greedy Heaviest Observed Sub-tree) protocol. The author [39] categorizes the consensus protocols into two kinds the traditional ones (multi-Paxos and PBFT) and the blockchain consensus protocol (Bitcoin) and gives a comparison between them. The Casper CBC protocol [39] has features associated with both kind of consensus protocols: providing low overhead and asynchronous BFT safety. The paper presents detailed description of different parameters involved and the protocol’s working.

A number of research efforts are going on to explore the possibility of using Ethereum and its smart contract concept to serve as eliminating the need of centralized party and reducing the operational cost involved in various financial sectors.

The prominent topics being studied in this sector include the cost savings, prediction, managing techniques, possible security vulnerabilities and oracles (data feeds), some of which are described here briefly:

Cost Effectiveness

According to Schneider et al. [40], the usage of blockchain-based technology can result in significant cost savings in the financial industries. In addition, the authors estimated that application of blockchain technology to the clearing and settlement process of cash securities could result in an annual savings of US \$ 11-14 B. Moreover, reduction in title insurance premiums and annual savings of US \$ 2-4 B can be achieved using blockchain in the title insurance process.

Another effort was made by Moyano and Ross [41] to study the application and benefits of the distributed ledger technology to the Know-Your-Customer (KYC) process carried out by the financial institutions. They suggested a system to reduce the inefficiencies and costs involved in the KYC process carried out by the financial institutions. In their proposed model, the redundancy of this process by different financial institutions is suggested to be removed. In their proposed model, if a client wants to be involved with multiple financial institutions, the KYC process is only carried out by one financial system and shared with the other institutions through the distributed ledger technology.

Managing Techniques

Muller et al. [10] presented a management system for financial contracts through which the execution of the contracts can be automated on a blockchain. For repre-

senting the financial contracts, they utilized the modified form of the language by Bahr et al. [42] and showed examples of different contracts designed using this language. These examples include forward contract, barrier option contract, to name a few.

Prediction Market

Bentov et al. [43] talked about the prediction market in which people gamble on the results of possible future events. They presented an idea of possible enhancements in the cryptocurrency to accommodate fully decentralized prediction market. For this, they employed a novel modification of concept of Colored coins [44]. In their work, they proposed that this concept can be extended to incorporate other financial instruments and have real-time exchange of cryptocurrencies without relying on the trusted third party.

Oracles

Tools to fetch information about financial data include Oraclize [45] and Schelling coin [46]. Private and public instances of different blockchains can be combined with Oraclize. Oraclize [45] is constantly monitoring the Ethereum blockchain for requests to fetch data. When it is received, Oraclize asynchronously fetches the results based on the parameters specified and sends the results in a transaction.

Eskandari et al. [47] developed a tool called *pricegeth* to fetch price and implemented a smart contract representing a customized derivative option. They deployed this contract on Ethereum. This tool is being used in my research. *Pricegeth*, as highlighted by the authors [47] fetches external information about exchange prices.

This is different from the other price fetching tools, such as *Oraclize*, as it allows user to access historical data. The *pricegeth* server continuously watches the *Ethereum testnet* to see if new blocks are added and whenever a new block is added, *pricegeth* fetches the exchange prices at that time [47].

Adler et al. [48] highlighted an issue with the *oracles* stating that these could serve as a centralized point of failure. The possibility of this issue is due to the fact that the *oracles* do not accompany strong security guarantees. As an alternative, the authors [48] proposed a decentralized oracle *Astrea* that is based on a voting game for determining the truthfulness of a proposition. There are three types of participants in the system: voters, submitters and certifiers. The propositions are issued by the submitter by paying some fees. The voters have a small risk and participate by placing small stake next to a random proposition. Certifiers face more risk and place higher stake during the process of voting and certification. The participants can enter and leave the system on their own will. In other words, they are not required to be online all the time. In case the certifiers and voters results match, the participants who voted towards it get rewarded and others penalized. In the other situation where their results don't match, certifiers are penalized. The authors [48] based on some experiments also highlight the value of parameters that can be set to reduce the adversary impact.

Possible Vulnerabilities

Another important factor to consider while developing the smart contracts is its security. In other words, the possible vulnerabilities present in the contract that might

be exploited by the adversaries for their gain. Bhargavan et al. [9] made an attempt to verify the smart contracts formally using a functional programming language called F^* . In their proposed approach, functional precision and run-time security of the Ethereum smart contracts can be studied.

A similar approach was adopted by Luu et al. [15] in which they tried to formalize the semantics for smart contracts. Interestingly, they were able to identify certain subtle concepts that helped them to design a vulnerability detection tool for smart contracts. The tool, called *Oyente*, which is based on symbolic execution and identifies several possible issues. When this tool was run on the existing contracts on Ethereum, around 45 percent were marked vulnerable.

Kalra et al. [49] came up with a formal verification framework known as ZEUS, which explores the smart contracts for vulnerabilities. It allows the verification of smart contracts through the use of abstract interpretation and symbolic checking with the clauses for constrains. It checks if the contracts follow safe programming methods and business logic that was consented [49]. The authors [49] claim that the framework has no false negative rate and that more than 90 percent of the contracts the framework was studied for were vulnerable.

Another scalable, public and automated analyzer was developed by Tsankov et al. [50]. This verifier allows the users to check if the way the contract behaves is safe or not with respect to any specific property. There are two types of patterns that the analyzer checks for making sure if the particular property holds. These are compliance and violation patterns and written in delegated domain-specific language. The compliance pattern suggests that a certain security property is fulfilled whereas

the violation property means that the security property is not satisfied. This analyzer significantly reduces manual effort by categorizing the behaviors into true violations and warnings (that user checks if they are true or not) [50]. The analyzer starts by taking the input which is either EVM bytecode or solidity contract (it converts it into bytecode). Then, it converts this bytecode into a stackless form, which is further checked for dependencies in data and control-flow. The semantic facts are stored in a system called *Datalog*. Further, the compliance and violation patterns are checked. If any successful violation pattern is found, the instruction causing it is presented as output. The authors [50] highlight a number of limitations of this system and how these are addressed.

Amani et al. [51] proposed a bytecode level program logic addition to the present EVM formalization. This extension is proposed in the Isabelle/HOL [52] that includes different correctness properties of smart contracts and using the concept of *gas* for a different way of handling termination. Isabelle/HOL [52] is the higher-order logic encoder in the theorem prover framework known as Isabelle. In their work [51], the authors used a pre-existing EVM formal model and Isabelle/HOL framework along with designing a program logic for the verification of EVM bytecode. Another contribution of authors [51] involve proposing a method in Isabelle to automatically generate verification constraints based on the program logic. The author's [51] idea has been accepted by the Ethereum foundation in their official repository for EVM formalization.

2.5.2 Hyperledger

Blockchain, as stated before, is an immutable chain of blocks comprising of transactions. This chain is maintained by the network of nodes called miners. The introduction of this technology was first done in the Bitcoin cryptocurrency where it was used as an underlying technology for storing transactions. An additional smart contract concept to encode any business rules was introduced in Ethereum blockchain. Through this smart contract concept, people can develop decentralized applications and define their rules in the form of a smart contract, which gets automatically executed every time the different functionalities of the contract are called. Although, the Ethereum blockchain is gaining a lot of attention, some of the institutions are still not that comfortable using it or putting their data on it. The main concern is that these blockchains are permissionless. In other words, anyone can start maintaining these blockchains and see the previous transactions and data. However, in some enterprises there is a concern of making their data open to the customers and there is a need to identify the users [53], which is not possible in the public blockchains.

Hyperledger [53], an open-source project, on the other hand, is a permissioned blockchain that is designed to facilitate blockchains in industries. In other words, the nodes maintaining the network and the ledger are selected through a Membership Service Provider (MSP) [53]. Hyperledger project comprises of various frameworks and tools that can be used for various purposes ranging from viewing data on blockchain to developing applications. There are various frameworks and tools present in Hyperledger. One of these projects that is being studied in my thesis is Hyperledger *fabric*. In this project, a node can have various roles, such as peer node, orderer and client.

A client as the name indicates play the role of the end-user and sends transactions to query the ledger. The peer nodes are the ones that keep the copy of ledger and maintain it. Some of these peers can be endorsers that is, they need to endorse a transaction before it is confirmed. Last but not the least, the ordering node determines the order of the transactions in the block and broadcasts this order to all the peer nodes so they have uniformity in the order of transactions in the block. Various steps included in the transaction life cycle are as follows [53]:

- The end-user or client initiates a transaction. The transaction proposal is created using one of the APIs available in the application layer. This proposal is sent to the target peers and contains details about the methods of *chaincode* that are to be invoked.
- The endorsing peers verify different details of the transaction proposal to see if the transaction is valid. In case of valid proposal, the peers execute the respective chaincode and produce a response for the request. The response along with endorsing peer's signature is sent back to the application.
- The application receives different proposals and verify if they are same and also the signature of the endorsing peers. Now, in case of an update request, the application makes sure if the endorsement policy is satisfied and then broadcasts the transaction proposal and response to the ordering peer.
- The ordering service collects and orders different transactions for one channel into a block.

- Then, these blocks are sent to the peers again. The peers confirm the endorsement policies fulfillment. The peers again check the validity and respective versioning of the transactions and blockchain. Transactions are marked valid or invalid.
- The peers then append these blocks to their ledger. The notification is sent to the clients.

The two main constituents of the ledger in Hyperledger fabric [53] are the *world state* and the *transaction log*. The transaction log comprises of all the transactions that have been incurred in the ledger that changes its state, whereas the *world state* depicts the present state of the ledger. Each of the entry in the world state is represented as a key-value pair. The smart contracts in Hyperledger fabric are known by the terminology of chaincodes. These define the rules for execution and are executed when a transaction is invoked. The languages that can be used to write the chaincodes include Java, Go, and Nodejs. Although the participants in the Hyperledger know each other, still their might be a lack of trust. Hence, Hyperledger fabric allows to incorporate different consensus protocols, such as *Kafta* or *solo*, to have a unified copy of ledger across all participants. Hyperledger does not have any native currency unlike other public blockchains but, users can create their own tokens if they wish to. As mentioned above the state of the database can be queried through a transaction. When the user sends a transaction it goes through various steps before updating the ledger.

There are six essential elements of Hyperledger fabric as mentioned in the Hyperledger fabric documentation [53]. These include assets, chaincode, ledger features,

privacy, security and membership services and consensus. Each of these is briefly described below:

1. **Assets:** The assets could represent anything, tangible or intangible. These are depicted using the key-value pairs where key has to be unique for any asset being stored on the blockchain.
2. **Consensus:** Consensus is required so that all participants have same series of blocks in the blockchain. Consensus also requires that the block and its transactions meet the policies defined. In Hyperledger, the process of consensus is involved throughout the transaction life cycle. This is a multi-stage process ranging from the endorsement policies to validation checks to identity verification and finally transaction versioning checks.
3. **Chaincode:** Chaincode, as mentioned before, is a set of rules that are encoded inside the ledger and executed on receiving a transaction. These basically define the logic of the business. The reading of and writing to the ledger is determined through these rules.
4. **Privacy:** Hyperledger has a concept of channels which has its own ledger and smart contracts. Only members that are part of this channel can update the ledger and maintain a copy of the ledger. These nodes are decided by the MSP. If the data is to be stored on a subset of these members, the concept of private data collection can be used instead of creating a new channel. The data that is to be stored is spread using the *Gossip* protocol. However, the hash of this data is stored on each and every peer who is a part of the channel.

5. **Security and Membership services:** The Hyperledger network consists of the members who have known identities and can be traced back. Cryptographic certificates are provided to different participants in the network. These are issued by the Public Key Infrastructures and determine the roles and access that the member has.

6. **Ledger features:** The ledger as mentioned before consists of a blockchain (transaction log) and the current state database. All the peers that are part of a channel maintain a copy of the ledger for that channel. The policies and access control lists (ACLs) are defined in the configuration file of the ledger. The clients can send transactions to query or update the ledger. The transactions include the versions that are used during versioning checks in the consensus process. The databases available for the ledger include LevelDB and CouchDB. The default DB is LevelDB, however it doesn't allow rich queries to access the database for which CouchDB is utilized.

Different organizations can use the Hyperledger technology to interact with each other and create applications. These applications depict the company's business logic and how they intend to incorporate the hyperledger into their business. For this, the organizations create a network that consists of ledger, chaincode and various configuration details such as access policies. The network is decentralized as it is formed by multiple organizations. The basic components and workflow of creating a network, as per the documentation [53], include the following:

- The network is initialized by an organization by starting an ordering service and defining the network configurations.

- A Certificate Authority (CA) is created that provides the different network nodes with certificates to interact with the blockchain or for endorsements. Hyperledger has a built-in CA known as *Fabric-CA* but organizations usually prefer their own CAs. MSPs determine the mapping between the certificates and the authorities.
- A channel is created through which different organizations can interact with each other.
- The peer nodes can be added to the channel that maintain the ledger. The peer is bound to the network through a certificate. The peer sends the request to the orderer to join the chain, which then checks the network configuration to see the permissions of this peer.
- Client applications can interact with the ledger and invoke the chaincode by sending transactions.

Various actors including peers, applications, orderers, etc., holds a certificate that represents their identity. This determines the permissions that these actors have in the blockchain network [53]. The certificate that defines these permissions is called X.509 digital certificate. The two main components that deal with the identity are PKI (certificate authority is part of this) and MSPs [53]. Having discussed the important and prominent blockchain platforms, I present in the next chapter some general work in creating smart contracts for these platforms not necessarily related to financial smart concepts. However, many of the key concepts in these works could be used for studying financial smart contracts.

Chapter 3

Related Work and Problem

Statement

The blockchain is being researched for its design and applicability to different industries. There are various blockchain platforms proposed in the past having different features. Some of these include Ethereum, Hyperledger, Quorum, R3 Corda, Ripple and IOTA. Ripple, Ethereum and Hyperledger have been discussed in detail in Section 2.4.4, Chapter 2.5.1 and Chapter 2.5.2 respectively. Quorum [54] is a permissioned blockchain and is based on the Ethereum protocol. Quorum blockchain is a fork of go-ethereum [55]. Quorum is an enhanced version of Ethereum blockchain that offers better performance than public *geth*. Quorum [54] features both public and private transactions/smart contracts and supports consensus protocols such as Raft-based Consensus (default) and Istanbul BFT. Quorum [54] is designed by reusing most of the design features of Ethereum in order to sync easily with the future releases of go-ethereum. R3 Corda [56] is an open source blockchain platform also offering an

enterprise version for businesses. Corda [56] enables secure and direct transactions between businesses through smart contracts. The main goals of the Corda project are to reduce time and cost of financial services. The Corda platform [57] is designed for maintaining records and processing of financial agreements that excludes the financial incompatible features present in other blockchain. The main constituents of consensus in this blockchain [57], is transaction validity (proposed transaction generates valid state and has been signed by required authority) and transaction uniqueness (similar transaction has not been produced before). IOTA [58] is another blockchain platform that offers fee-free microtransactions. In IOTA platform, each proposed transaction is linked together unlike the other blockchain platforms where transactions are collected into blocks that are linked together. The consensus protocol in this involves the participant to verify two previous transactions and as a reward get their transaction validated by some other peer resulting into entangled transactions.

These blockchain platforms are being studied for applications in different industries, one of them being finance industry for achieving decentralization or transparency, reducing costs and processing time.

Eskandari et al. [47] developed a tool called *pricegeth* to fetch price and implemented a smart contract representing a customized derivative option. They deployed this contract on Ethereum. They adjusted their contract and made it fully collateralized [47] along with setting a limit on the loss incurred by the participants in the contract. My idea of representing an option contract is similar to the one presented in [47]. To the best of my knowledge, there are no work reported in the literature for financial collateral service smart contracts using blockchain technology. *Pricegeth*, as

highlighted by the authors in [47] fetches external information about exchange prices. This is different from the other price fetching tools, such as Oraclize, as it allows user to access historical data. The *pricegeth* server continuously watches the Ethereum testnet to see if new blocks are added and whenever a new block is added, *pricegeth* fetches the exchange prices at that time.

The Depository Trust and Clearing Corporation (DTCC) [59] collaborated with IBM, Axoni and R3 to design a distributed platform for post trading process to automate and remove the requirement of disjoint and repeated processing in order to reduce costs. The industry planned [59] to redesign its existing Trade Information Warehouse (TIW) into a blockchain based platform with IBM selected to lead the initiative. Axoni and R3 were chosen to provide the ledger infrastructure/contract applications and solution advise respectively. The resultant distributed ledger platform will include having peer nodes at the different firms involved and the network being governed by DTCC. Various industries that provided input to this solution include J.P. Morgan, UBS, Barclays, etc.

Problem Statement All these studies provide general design considerations for smart contract but not necessarily for financial smart contracts. For my thesis, I am focusing on designing transparent, decentralized and cost effective financial smart contracts on blockchain.

My methodology in achieving my goals is presented in the next chapter. The result of the thesis could be used as guidelines for setting up Blockchain platform(s) for Collateral Smart Contract Services.

As mentioned before, to the best of my knowledge, there are no works reported in

the literature for financial smart contracts using Blockchain technology. It is however, possible that there are such implementations in financial institutions as proof-of-concept or even for practical use.

Chapter 4

Solution Methodology

My methodology is divided into two sections: in the first section the one used for implementation on Ethereum is discussed and in the second section the one used for Collateral Contract Services on Hyperledger is explained.

4.1 Methodology adopted for Options Contract on Ethereum

I describe my methodology in the following two sections explaining creation of smart contracts and analyzing the smart contracts for vulnerabilities.

4.1.1 Creation of Smart Contracts

My approach differs from a recent work [47] in that the contract created by them is a currency swap option whereas my approach is the actual financial option traded in finance market. In the contract, the functionality for both put and call options are

included. In this section, I describe how a smart contract is created.

An option is a derivative whose value depends on the underlying asset, which in our case is the Bitcoin to Ether (BTCETH) exchange rate. However, at present the exchange data is not on the Ethereum blockchain, hence there is a need to fetch the data from external sources. This data will decide the actual transfer between the holder and the writer at the expiration time. Therefore, there is a dependency on information feed that fetches information on the market conditions. For this purpose, I am using *pricegeth* [47]. It is an Application Programming Interface (API) on Ethereum blockchain [60]. The server of *pricegeth* constantly monitors the Ethereum Testnet called Ropsten. Whenever it observes that a new block is added, the “Poloniex Prices (exchange rate)” at that time are saved by the server. Poloniex [61] is the exchange API used to fetch the exchange rate. Entire historical prices are stored by the *pricegeth*. These prices are further sent to the *pricegeth* smart contract, which can be queried by other contracts for price at specific block number [60].

Environmental Setup: For implementing the smart contract for instance, for a simple European Option, I am using the language called *Solidity* [62]. *Solidity* is a high-level language that is a lot similar to Javascript and is influenced by different languages including Javascript, Python and C++ [62]. This language is contract oriented and user-friendly. After the smart contracts are written, they are compiled into EVM bytecode and published on the blockchain. As to write any program, a debugger is required. For this purpose, I am using Remix [63] Integrated Development Environment (IDE). Remix is an IDE with the main concentration on the development and testing by deployment of the smart contracts written in *Solidity* language. This

IDE provides different useful features, which includes debugging and developing a smart contract, analysis of the code being written and warning the user about possible flaws, debugging any committed transaction, etc. Using this IDE along with *mist* and *geth*, allows user to verify and debug the applications being developed [64].

Algorithms 1-3 presents the pseudo-code for smart contract:

The *buyCallOption* function can be triggered by the users to enter into the contract as a buyer of the call option. In this case, the contract will serve as the writer in the option. To obtain these rights, the buyer has to send some ether with the transaction that will be utilized to pay the premium to the writer. This premium has to be equal to the amount of premium required for the option contract. On the other hand, the writer (contract itself) has to make sure that it has enough balance (named, locking amount) to transfer to the holder as profit. This amount will be used in the future to clear the option, in case it is exercised.

Algorithm 1: *buyCallOption* method

Input: Sender's (Holder) Address, premium**Output:** Allocated optionId**Required:** $Premium\ Sent = Option\ Premium\ Required$;**Required:** $Contract's\ balance = Default\ Settlement\ Balance$;**if** *Conditions Required Satisfied* **then**

| Option Created and Initialized;

| Premium is paid to the writer;

| returns Option ID;

end**else**

| Premium is refunded;

| Execution is not successful;

end

Instead of taking the holder's position, the user can execute the *writeCallOption* function of the contract (Algorithm 2). In this option, the user will act as the writer whereas the smart contract will act as the holder of the option. In this scenario, the contract must have enough balance to pay the premium to the user (writer). Also, the user must send enough money that will be used later to clear the account.

Algorithm 2: *writeCallOption* method

Input: Sender's (Writer) Address, Collateral Amount**Output:** Allocated optionId**Required:** *Collateral Amount* \geq *Min. Default Balance* ;**Required:** *Contract's balance* = *Premium Required*;**if** *Conditions Required Satisfied* **then**

| Option Created and Initialized;

| Premium is paid to the writer;

| returns Option ID

end**else**

| Default balance is refunded;

| Execution is not successful;

end

At the expiration time (in terms of block number), the *exerciseOption* function (Algorithm 3) can be called through a transaction. Depending on the exchange price at that block number and the type of option, the default balance amount locked in the beginning will be used to transfer as profit to the appropriate account.

Algorithm 3: *Exercise Option*

Input: Sender's Address**Output:** Bool: Success/Fail**Required:** *Valid Sender Option ID ;***Required:** *Contract not exercised before;***Required:** *Valid Exercise Time;***if** *Current Stock Price == 0* **then**

| Return False

end**else if** *Call Option* **then**| **if** *Stock price > Strike price* **then**

| | Transfer profit and adjust balance

| **end**| **else**

| | Adjust balance

| **end****end****else**| **if** *Strike price > Stock price* **then**

| | Transfer profit and adjust balance

| **end**| **else**

| | Adjust balance

| **end****end**

4.1.2 Analyzing smart contracts

I analyze the smart contract created for potential vulnerabilities. The main idea is to identify the bugs in the smart contracts before they are deployed on the blockchain. This is highly essential due to the fact that once a smart contract is published without proper analysis, the results can be catastrophic. It has happened once in the case of the famous Decentralized Autonomous Organization [65] attack in 2016, with the consequence of around \$50 million worth of ether being lost.

The tool that I am using for this purpose is *Oyente* [15]. It is an open-source tool available on github. There exist various strategies to implement this tool. One can implement a quick version using the docker implementation or full installation.

The vulnerabilities that *Oyente* [15] analyzes in a contract are as follows:

(a) Transaction - Ordering Dependency; (b) Time Stamp Dependency; (c) Mishandled Exceptions; (d) Re-entrance Vulnerability; and (e) Integer Overflow/ Underflow

I have studied the possibility of these vulnerabilities in the smart contract that I created.

4.2 Methodology adopted for implementation on Hyperledger

The main approach of having automation and transparency in the collateral services process is by recording the day-to-day collateral exchange transactions on a distributed ledger. As a Proof-of-Concept, I have an Organization (FinBank) who has to exchange collateral with two different Organizations (TechFin and FinIns) for

covering the margin calls.

Environmental Setup: There are certain pre-requisites that need to be installed for experimenting with Hyperledger Fabric blockchain as indicated in the Hyperledger docs [53]. These includes *Docker Toolbox* for Windows, *GitBash*, *Node.js*, *Go* programming language, *curl*, and *python*. Docker serves as an environment for all the executions. The existing Hyperledger binaries that are required for setting up the network are downloaded on the system using the *curl* command. In Hyperledger, the smart contracts are known with the term *Chaincode*, which can be written using different languages such as Node.js, Java and *Go*. In my thesis, I use *Go* programming language to write this chaincode. This chaincode defines the rules for storing the company's assets on the Blockchain and issuing them as collateral to pay off the margin-calls. The basic functions included in this chaincode include creating, sending, receiving a collateral, querying a collateral, and querying the history of a collateral. The company's assets are stored in the blockchain as key-value pairs.

As a first step, the network (named *collateralNetwork*) is setup consisting of three Organizations named *FinBank*, *TechFin* and *FinIns*. There exists one peer per organization that maintains the copy of the ledger and a single orderer organization for the entire network. Another important factor during network setup is that the collateral transactions between *FinBank* and *TechFin* should not be visible to *FinIns* and likewise transactions between *FinBank* and *FinIns* should not be open to *TechFin*. There are two possible ways of achieving this by creating two different channels or through private data concept. In my thesis, I introduce different channels to achieve this required privacy. The current implementation consists of two channels showing

how one organization is handling its relationships with two different companies as shown in Figure 4.1.

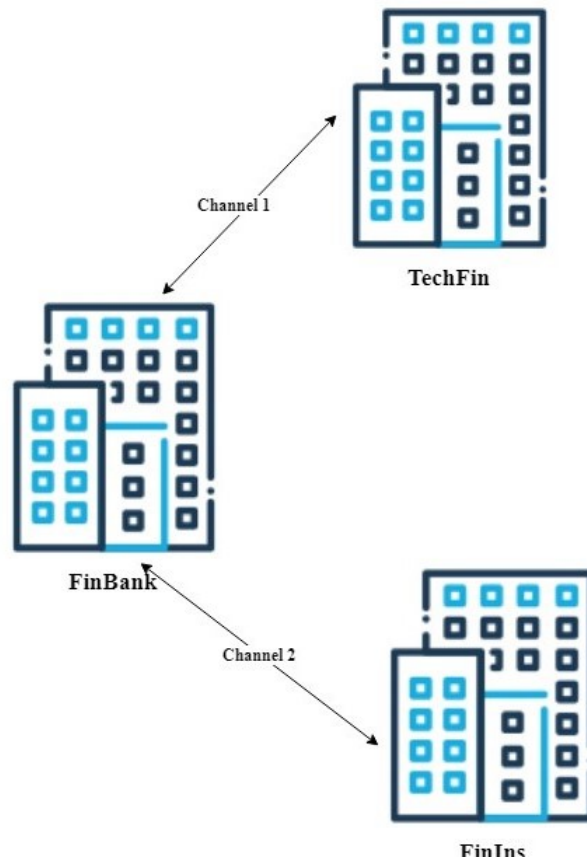


Figure 4.1: Collateral Blockchain Network

For storing the assets on the blockchain, I used CouchDB that permits complex queries against its data. For each peer, an instance of CouchDB is created. The ordering mechanism to order transactions used in this network is called *solo*. The rules of various functions on a collateral are defined using Chaincode.

The various steps followed for setting up the network are as follows:

1. The details about the network and various entities involved is included in a *yaml* config file. The cryptogen tool that was downloaded previously is used to create essential signing keys and X509 certificates for these entities. The network mocks the traditional network, where each member organization have its own Certificate Authority, by assigning a unique certificate to organizations.
2. Then the configuration files or artifacts are generated using the configtxgen tool downloaded before. These artifacts include the orderer genesis block (that includes root certificate details for organizations added), channel configurations and anchor peers. The input to this tool is a *yaml* file that defines the details of network including anchor peers and channels.
3. Then I start up the network using the docker-compose.yaml file that launches various containers required for different components including peers, CAs, orderer node (with genesis block), couchDB for each peer and a cli container.
4. Channels are created using the artifacts generated earlier and different peers are joined to the channels. Also, the channels are updated to incorporate the definition of the anchor peers for the organizations.
5. The chaincode is installed on the peers and instantiated on the channel while defining required endorsement policies. When peers interact with the chaincode, a chaincode container is created.

For interacting with the network, an application is developed using Hyperledger Fabric Software Development Kit for Node.js. The SDKs are used to connect the application to the fabric network and send transactions to invoke different functions

of the chaincode. These transactions can either be used to query or update the ledger. The collateral network and its components are shown in Figure 4.2.

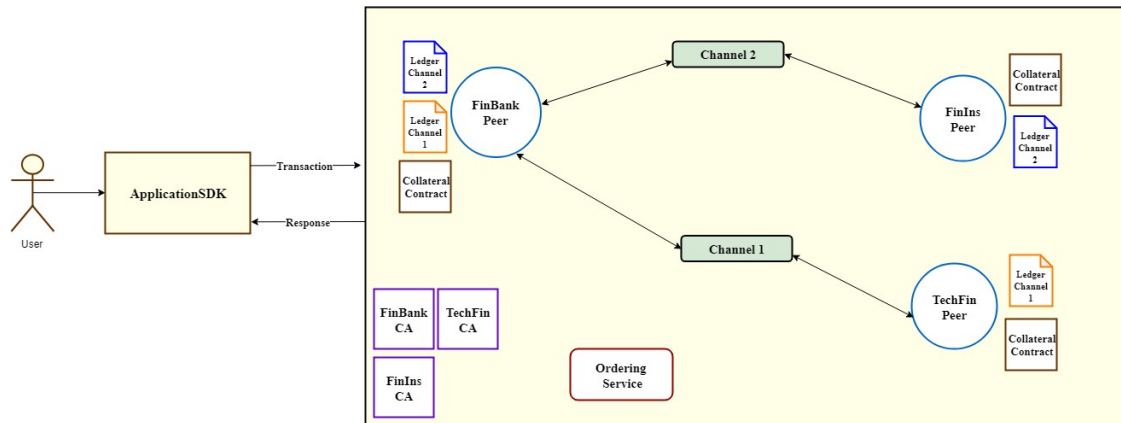


Figure 4.2: Collateral Blockchain Network Components

When the network is setup, a default administrator user (admin) is registered with each of the CA servers that can be used to register other users. Then, the Certificate Signing Request (CSR) is used to get certificate for the admin that allows the admin to act as administrator for that CA. Another user is created for each Organization (Alice for FinBank, Manet for TechFin and John for FinIns) that is used to interact with the blockchain through Node.js application. Different transactions are submitted by connecting to the gateway and accessing the contract of specific channel.

A larger network has been setup as well as shown in Figure 4.3, however the experiments and results are from this small scale network. As FinBank peer is part of all channels so it will contain a ledger from each channel.

Once the environment is setup, it is ready to be experimented with and the next Chapter explains my experiments and the outcome of these experiments in achieving the goals set at the beginning in creating collateral services contracts.

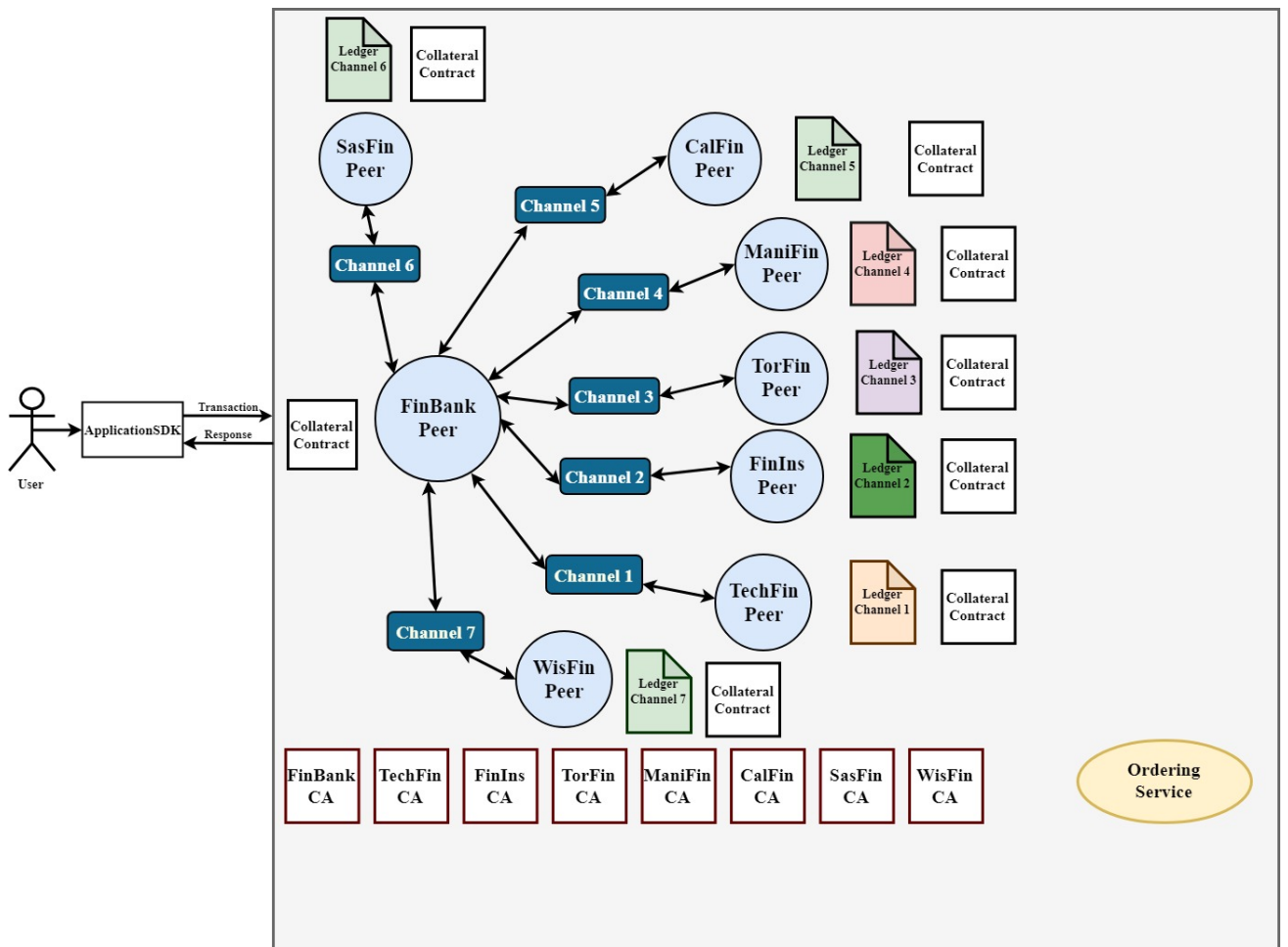


Figure 4.3: Large Scale Collateral Blockchain Network

Chapter 5

Experiments and Results

This chapter has two sections where first one shows the experiments and results from initial approach of designing application for options on Ethereum blockchain and second section for Collateral Contract Services using Hyperledger.

The goals of my experimental evaluations in public/private blockchains are as follows:

- For Ethereum:
 - Creating a financial smart contract for Options with different scenarios such as buying/selling, profit and time
 - Deploying it on the Ethereum Environment using Remix IDE
 - Testing the deployment utilizing user wallets on JavaScript VM environment
 - Identifying the vulnerabilities using an open source tool called Oyente
- For Hyperledger:

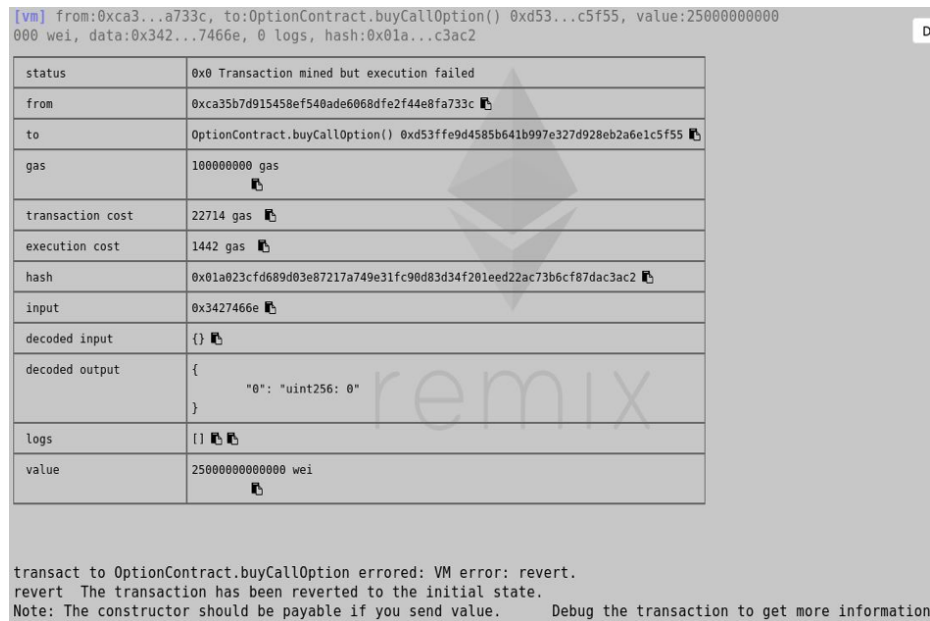
- Setting up an appropriate network of peers and other entities such as Certificate authorities, ordering services, etc.
- Setting up appropriate privacy among peers using channels
- Incorporate the collateral details on the ledger and sending it towards margin
- Executing transactions which have details about the collateral contract that gets added to a block (including the type of collateral, worth of collateral, sender, etc.)

5.1 Design outcomes of Smart Contract on Ethereum

I have designed and implemented a contract that contains several functions to enter into European call option. The functionality of the contract has been tested using the JavaScript Environment present in the Remix IDE. When the contract is executed, some ether are used from the account as gas for contract creation as shown in Figure 5.1.

The different parameters in the result shows details about the contract creation. The transaction cost tells what was the cost for sending the contract code to ethereum and the execution cost tells the cost for VM execution. The contract address tells about the address of the contract. The input shows the bytecode of the contract and the hash represents the transaction hash.

The transaction generated on the execution of *buyCallOption* function was mined but reverted as shown in Figure 5.2 due to the error that the contract did not have

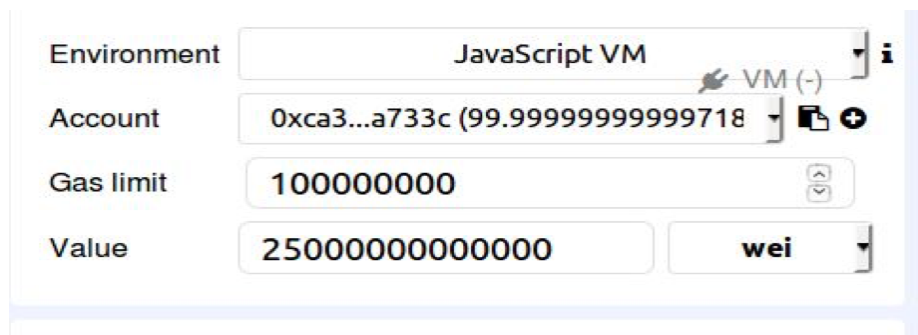


status	0x0 Transaction mined but execution failed
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	OptionContract.buyCallOption() 0xd53ffe9d4585b641b997e327d928eb2a6e1c5f55
gas	100000000 gas
transaction cost	22714 gas
execution cost	1442 gas
hash	0x01a023cfd689d03e87217a749e31fc90d83d34f201eed22ac73b6cf87dac3ac2
input	0x3427466e
decoded input	{}
decoded output	{ "0": "uint256: 0" }
logs	[]
value	250000000000000 wei

transact to OptionContract.buyCallOption errored: VM error: revert.
revert The transaction has been reverted to the initial state.
Note: The constructor should be payable if you send value. Debug the transaction to get more information

Figure 5.2: BuyCallOption - error

successfully as shown in Figure 5.4.



Environment: JavaScript VM

Account: 0xca3...a733c (99.99999999999718)

Gas limit: 100000000

Value: 250000000000000 wei

Figure 5.3: BuyCallOption

If the function `buyCallOption` is executed without passing any value, then the transaction fails because the premium has to be paid by the buyer.

If the sender wants to be a writer of the call option, then sender can call the `writeCallOption()` function. On calling this function, with the input as the amount of *wei* equal to the call options default value to be used for settlement at the expiration

status	0x1 Transaction mined and execution succeed
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	OptionContract.buyCallOption() 0xd53ffe9d4585b641b997e327d928eb2a6e1c5f55
gas	100000000 gas
transaction cost	222057 gas
execution cost	200785 gas
hash	0xc7ba2aaca1372b90f74cf6265d7624e32f3721b37dd1c87f861af2cd3bc57d8f
input	0x3427466e
decoded input	{}
decoded output	{ "0": "uint256: 1" }
logs	[{ "topic": "421c339eab8cf6de2ee2fb87270af7586192e6c994e25350b583518c4876b7bf", "event": "PremiumPaid", "args": ["1", "0xd53ffe9d4585b641b997e327d928eb2a6e1c5f55", "25000000000000", "Premium Paid to the receiver"] }, { "topic": "96561394bac381230de4649200e8831afcab1f451881bbade9ef209f6dd30480", "event": "LogMessage", "args": ["new Option has been created with the received parameters"] }, { "topic": "01c3b3e818499a7d10242fd5f454233f0d78b6de84c98bc590ae07e29bb757c0", "event": "LogOption", "args": ["1", "0xca35b7d915458ef540ade6068dfe2f44e8fa733c", "25000000000000", "1150004", "Buy Call Option"] }]
logs	[{ "topic": "421c339eab8cf6de2ee2fb87270af7586192e6c994e25350b583518c4876b7bf", "event": "PremiumPaid", "args": ["1", "0xd53ffe9d4585b641b997e327d928eb2a6e1c5f55", "25000000000000", "Premium Paid to the receiver"] }, { "topic": "96561394bac381230de4649200e8831afcab1f451881bbade9ef209f6dd30480", "event": "LogMessage", "args": ["new Option has been created with the received parameters"] }, { "topic": "01c3b3e818499a7d10242fd5f454233f0d78b6de84c98bc590ae07e29bb757c0", "event": "LogOption", "args": ["1", "0xca35b7d915458ef540ade6068dfe2f44e8fa733c", "25000000000000", "1150004", "Buy Call Option"] }]
value	25000000000000 wei

Figure 5.4: Successful Transaction for Buy Call Option

date, the transaction was successful. If the sender is a writer of an option, then the premium will be sent to the sender from the contract.

Results from execution of Write Call Option Function is illustrated below in Fig-

ure 5.5

status	0x1 Transaction mined and execution succeed
from	0xca35b7d915458ef540ade6068dfc2f44e6fa733c
to	OptionContract.writeCallOption() 0x50c9cd0fe417858dccb05269e47abc572aa77
gas	10000000 gas
transaction cost	231868 gas
execution cost	218596 gas
hash	0x603ee531e673c8fe9aa29f9e1966f97842958108705573f94764035cc7f85
input	0x76886794
decoded input	{}
decoded output	{ "0": "uint256: 1" }
logs	[{ "topic": "421c339eab8cf6de2e2fb8270af7586392e6c994e25350b58318c4876b7bf", "event": "PremiumPaid", "args": ["1", "0xca35b7d915458ef540ade6068dfc2f44e6fa733c", "25000000000000", "Premium Paid to the receiver"] }, { "topic": "96561394ba381230d4649208e8831afcab1f451881bbade9ef209f6d030480", "event": "LogMessage", "args": ["new Option has been created with the received parameters"] }, { "topic": "01c3b3e818499a7d10242d5f454233f0d78b6de84c98bc598ae07c29b6757c0", "event": "LogOption", "args": ["1", "0xca35b7d915458ef540ade6068dfc2f44e6fa733c", "100000000000000", "1150000", "Write Call Option"] }, { "topic": "79d3318da17d03032c878a80259d4d4e6707288e1f8fa49c6e67610c7448172", "event": "ExerciseMsg", "args": ["The contract can be exercised after specified number of blocks in variable exerciseTime have been added after this block"] }]
value	100000000000000 wei

Figure 5.5: Successful Transaction for Write Call Option

The contract designed was deployed on the Ethereum Test network (Ropsten Testnet). The various steps followed for deploying the contract on Ethereum are shown in Figure 5.6

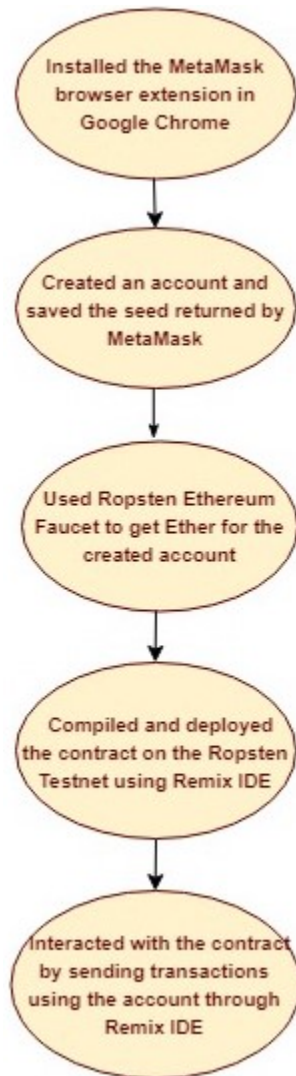


Figure 5.6: Steps for deployment on Ethereum Ropsten Testnet

The results generated using Oyente [15] for the initial and the final version are depicted in Table 5.1.

In the initial version of the contract developed, default values were set for the price at expiration time without actually fetching data from real market. However, in the final version, the contract uses *pricegeth* to fetch the feed i.e. the price at the expiration time.

Most of the vulnerabilities remained even when address of the pricegeth contract was used instead of default values. However, with the default values the code coverage is slightly higher (EVM code coverage basically means how many opcodes of the total opcodes present in the file were executed) and the Transaction-Ordering Dependency is introduced in case of using pricegeth contract details. Transaction-Ordering dependency vulnerability is possible when the order in which the related transactions (calling same smart contract) are placed in a block impacts the results. This can be used to gain benefits by a malicious user.

Table 5.1: Vulnerabilities identified using Oyente

Vulnerability Type	Default Values	Using pricegeth
EVM Code Coverage	45.8%	44.7%
Integer Underflow	True	True
Integer Overflow	True	True
Parity Multisig Bug 2	False	False
Callstack Depth Attack Vulnerability	False	False
Transaction-Ordering Dependence (TOD)	False	True
Time-stamp Dependency	False	False
Re-Entrance Vulnerability	False	False

During the experiments, the pricegeth was not working to fetch current data as its functionality was impacted due to an attack on Ethereum. As a solution, hard-coded values and some of the historical data obtained from the pricegeth were used.

There are number of issues that an organization, which prefers to use Ethereum public blockchain needs to be concerned about such as the identity of the users being anonymous, the data is visible to everyone in the network, and possible attacks on Ethereum from time to time that could lead loss of *Ether*.

5.2 Design Outcomes of Smart Contract on Hyperledger blockchain

As mentioned in Section 4.2, three organizations were set up that exchange collateral contracts. The contract created has been tested for various functionalities through end user interaction through SDK. The result of starting up the network entities containers is shown in Figure 5.7:

```
Creating network "collateralnetwork_collateralNet" with the default driver
Creating ca.finbank.ordererN.com ... done
Creating ca.finins.ordererN.com ... done
Creating couchdb2 ... done
Creating ca.techfin.ordererN.com ... done
Creating couchdb1 ... done
Creating cli ... done
Creating ordererNode.ordererN.com ... done
Creating couchdb0 ... done
Creating peer0.techfin.ordererN.com ... done
Creating peer0.finins.ordererN.com ... done
Creating peer0.finbank.ordererN.com ... done
```

Figure 5.7: Collateral Network Setup

There is a container for a peer of each organization with different properties. There are different couchDB instance for each of the peers. Two channels are created channel 1 and channel 2. Peers of FinBank and TechFin are joined to channel 1 and peers of FinBank and FinIns are joined channel 2. The chaincode collateral is installed on peers and instantiated on the channel.

When the network is setup a default identity called “admin” is registered with the CA of each organization. In the next step, this default identity “admin” is enrolled for the FinBank, FinIns and Techfin as shown in Figures 5.8, 5.9, and 5.10, so that other users can be registered through this identity.

```
muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication/Finbank ((v1.4.0))
$ node enrollFinbankAdmin.js
Ca url : http://192.168.99.100:7054
path to wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\Finbank\wallet
Successfully enrolled the admin for Finbank

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication/Finbank ((v1.4.0))
$ node enrollFinbankUser.js
path to wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\Finbank\wallet
Successfully enrolled the alice user for Finbank
```

Figure 5.8: Successful Admin enrollment and user registration for FinBank

```
muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication/TechFin ((v1.4.0))
$ node enrollTechFinAdmin.js
Ca url : http://192.168.99.100:8054
Successfully enrolled the admin for TechFin
```

Figure 5.9: Successful Admin enrollment for TechFin

```
muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication/FinIns ((v1.4.0))
$ node enrollFinInsAdmin.js
Ca url : http://192.168.99.100:9054
path to wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinIns\wallet
Successfully enrolled the admin for FinIns
```

Figure 5.10: Successful Admin enrollment for FinIns

Using this administrator identity, one user is created for each of the organizations. Users Alice, Manet and John are registered for FinBank, TechFin and FinIns respectively, as depicted in Figures 5.8, 5.11 and 5.12.

```
muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication/TechFin ((v1.4.0))
$ node enrollTechFinUser.js
Successfully enrolled the manet user for TechFin
```

Figure 5.11: Successful User registration for TechFin

```

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication/FinIns (v1.4.0)
$ node enrollFinInsUser.js
path to wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinIns\wallet
Successfully enrolled the john user for FinIns

```

Figure 5.12: Successful User registration for FinIns

The above created users can interact with the ledger by sending transactions. Initially, the ledger does not have any collateral entry. So, firstly a collateral is created in the Hyperledger by the user Alice of Finbank organization on channel1 connecting FinBank and TechFin as shown in the Figure 5.13 To ensure that the collateral was successfully entered into the ledger, user Alice queries the collateral on channel1 by passing the ID of the collateral. The query returns the collateral details as shown in Figure 5.14.

```

$ node createCollateral.js --userName=alice --channelName=channel1 --orgName=FinBank --connectionFile=connectionFinbank.yaml
Connection File name is : connectionFinbank.yaml
Path to the wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinBank\wallet
2019-03-07T03:59:42.937Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "735340b5723533dfb7b53dc85ddc0cb677c9d0c8e72b3f5ccf324872ad6cc906"

```

Figure 5.13: Collateral Creation by Alice

```

$ node query.js --userName=alice --channelName=channel1 --orgName=FinBank --connectionFile=connectionFinbank.yaml
Connection File name is : connectionFinbank.yaml
Path to the wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinBank\wallet
Transaction has been evaluated, collateralresult is: {"collateralId":1291,"collateralName":"asset","collateralSender":"finbank","collateralworth":123.1,"currentState":"new","docType":"Collateral","issueDate":"2019-02-06T20:16:00Z","owner":"finbank"}

```

Figure 5.14: Collateral Query Result by Alice

The organization has to transfer an asset to the other organization named TechFin to cover the daily margin requirement. To achieve this, user Alice updates the state

of the newly created contract by calling `sendCollateral` method of the chaincode. This updates the current state and the owner organization of the collateral as can be seen in Figure 5.15.

```

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication ((v1.4.0))
$ node sendCollateral.js --userName=alice --channelName=channel1 --orgName=FinBank --connectionFile=connectionFinbank.yam1
Connection File name is : connectionFinbank.yam1
2019-03-07T04:18:03.127Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "af1cd8d3dd199736f7a78134236a0d692559f7784adc24155e719a93256f1868"

```

Figure 5.15: Collateral Used to pay margin

These changes can be observed on querying the ledger again as shown in Figure 5.16

```

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication ((v1.4.0))
$ node query.js --userName=alice --channelName=channel1 --orgName=FinBank --connectionFile=connectionFinbank.yam1
Connection File name is : connectionFinbank.yam1
Path to the wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinBank\wallet
Transaction has been evaluated, collateralresult is: {"collateralId":1291,"collateralName":"asset","collateralSender":"finbank","collateralWorth":123.1,"currentState":"paid","docType":"Collateral","issueDate":"2019-02-06T20:16:00Z","owner":"techfin"}

```

Figure 5.16: Collateral Query Result by Alice after paying collateral

Since user Manet is part of this channel, Manet can query the ledger and get the collateral list. The successful execution of query collateral result by Manet is shown in Figure 5.17. The query returns the current state of the ledger's entries.

```

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication ((v1.4.0))
$ node query.js --userName=manet --channelName=channel1 --orgName=TechFin --connectionFile=connectionTechFin.yam1
Connection File name is : connectionTechFin.yam1
Path to the wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\TechFin\wallet
Transaction has been evaluated, collateralresult is: {"collateralId":1291,"collateralName":"asset","collateralSender":"finbank","collateralWorth":123.1,"currentState":"paid","docType":"Collateral","issueDate":"2019-02-06T20:16:00Z","owner":"techfin"}

```

Figure 5.17: Collateral Query Result by Manet

As user Alice is a part of the second channel, i.e. the one connecting FinBank

and FinIns, as well, Alice can create a collateral on that channel too. The successful creation of collateral by Alice on the other channel2 is shown in Figure 5.18.

```

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication ((v1.4.0))
$ node createCollateral.js --userName=alice --channelName=channel2 --orgName=FinBank --connectionFile=connectionFinbank.yaml
Connection File name is : connectionFinbank.yaml
Path to the wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinBank\wallet
2019-03-07T04:23:23.040Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "830c26fa07398d87e6fefec5116fb2940a2ce7266fc2d0ce5e4a9279066d981b"

```

Figure 5.18: Collateral Creation Result by Alice on Channel 2

Being a part of the same channel, user John can query this collateral. The results of the query are shown in Figure 5.19

```

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication ((v1.4.0))
$ node query.js --userName=john --channelName=channel2 --orgName=FinIns --connectionFile=connectionFinIns.yaml
Connection File name is : connectionFinIns.yaml
Path to the wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinIns\wallet
Transaction has been evaluated, collateral result is: {"collateralId":129,"collateralName":"asset","collateralSender":"finbank","collateralWorth":123.5,"currentState":"new","docType":"collateral","issueDate":"2019-02-06T22:22:00Z","owner":"finbank"}

```

Figure 5.19: Collateral Query Result by John

If John tries to query on the channel 1 that he's not a part of, he gets an error as depicted in Figure 5.20

```

muskan@MSI MINGW64 ~/go/src/github.com/fabric-samples/collateralApplication ((v1.4.0))
$ node query.js --userName=john --channelName=channel1 --orgName=FinIns --connectionFile=connectionFinIns.yaml
Connection File name is : connectionFinIns.yaml
Path to the wallet is : C:\Users\muskan\go\src\github.com\fabric-samples\collateralApplication\FinIns\wallet
2019-03-07T05:29:18.914Z - error: [Client.js]: Channel not found for name channel1
2019-03-07T05:29:18.916Z - error: [Network]: _initializeInternalChannel: no suitable peers available to initialize from
The query transaction was not successful due to the error: Error: no suitable peers available to initialize from

```

Figure 5.20: Error produced in Query Result by John

The network and experiments designed for this thesis would encourage financial

institutions use Hyperledger to update their operation that can be visible to permissioned participants in the network that would help achieve transactions faster and seamlessly. Hence, these institutions may avoid having their own databases and maintaining them, and as well significantly reduce their work redundancy.

Chapter 6

Conclusion and Future Work

In my thesis, I have successfully designed financial smart contracts for collateral services in two different blockchain platforms, namely Ethereum (public) and Hyperledger Fabric (permissioned). The blockchain architecture can be deployed in any commercial institutions such as Banks.

Each of these platforms have their own advantages. Ethereum blockchain can be used to create decentralized applications on which the transactions are validated by miner node that could be any entity. In contrast, in Hyperledger Fabric the entities in the network are known, however, they might not trust each other. The users in the applications are permissioned and possess certificates that they utilize to interact with the Hyperledger Fabric network.

The programming language used to create smart contract in Ethereum is Solidity, which is still in its infancy and didn't support many features such as decimal data-type. On the other hand, Go programming language is used to create contracts on Hyperledger Fabric, which is well-developed.

The implementation on Hyperledger Fabric is more time-consuming as there are different elements that are required to get the network and chaincode running on the system unlike Ethereum, where a smart contract can be deployed to Ethereum testnet using Remix IDE.

The experimental evaluation criteria mentioned in Chapter 5 can be used as step-by-step guidelines for designing financial smart contracts on Ethereum and Hyperledger.

For the experimental results with Hyperledger Fabric network in this thesis, the number of institutions involved is limited to three. I have shown that it could be expanded to a larger network. Therefore, in future the network can be scaled up to include many other organizations and parties that could be involved in Collateral Contract Services. Also, a web user interface can be developed to facilitate participation of organizations in collateral services easily.

In Ethereum, there is still some sort of centralization in using Oracles to fetch external information. In future, some of these issues can be studied on how to achieve complete decentralization by studying other possible Oracles.

Application of blockchain technology has been promoted in various real world applications. With some effort, the current architecture used for financial smart contracts deployment on Ethereum and Hyperledger blockchains can be modified to study other types of applications such as supply chain management, food tracking, transportation, etc.

Relevant Publications

1. Managing Cryptocurrency Transactions with Blockchain Technology: A Status Survey, Muskan Vinayak, Saulo Santos, Ruppia K. Thulasiram, Parimala Thulasiraman, S.S. Appadoo, Presented at the Annual Conference of the Administrative Sciences Association of Canada, May 2018, Toronto, Canada.
2. Analyzing Financial Smart Contracts for Blockchain - Muskan Vinayak; Har Amrit Pal Singh Panesar; Saulo dos Santos; Ruppia K. Thulasiram; Parimala Thulasiraman; S.S. Appadoo - IEEE Cybermatics 2018 - International Conference on Blockchain, July 2018, Halifax, Canada.
3. A Novel Heuristics for Validating Pairwise Transactions on Cryptocurrencies - Saulo dos Santos; Daniyal Khowaja; Muskan Vinayak; Ruppia K. Thulasiram; Parimala Thulasiraman - IEEE Cybermatics 2018 - International Conference on Blockchain, July 2018, Halifax, Canada
4. Validating pairwise transactions on cryptocurrencies: a novel heuristics and network simulation, dos Santos, S., Vinayak, M., Thulasiram, R. K., Thulasiraman, P., Kamali, S. (2019), Springer Journal of Banking and Financial Technology, 1-11.

5. Designing Financial Smart Contracts for Blockchain - Muskan Vinayak, Ruppa K. Thulasiram, Saulo dos Santos, S.S. Appadoo - Hickson Research Day Poster Competition, Asper School of Business, University of Manitoba - To be held on April 5, 2019.

6. FSCBlock: Designing Financial Smart Contracts on Permissioned and Public Blockchains, Manuscript under preparation for submission (2019).

Bibliography

- [1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
- [2] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, “Karma: A secure economic framework for peer-to-peer resource sharing,” in *Workshop on Economics of Peer-to-Peer Systems*, vol. 35, 2003.
- [3] M. Bartoletti and L. Pompianu, “An empirical analysis of smart contracts: platforms, applications, and design patterns,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 494–509.
- [4] “White Paper : A Next-Generation Smart Contract and Decentralized Application Platform,” <https://github.com/ethereum/wiki/wiki/White-Paper>, online: Accessed; February 06, 2019.
- [5] J. Aron, “Bitcoin software finds new life,” 2012.
- [6] S. Sprankel, “Technical basis of digital currencies,” *Technische Universität Darmstadt*, 2013.
- [7] N. Szabo, “Formalizing and securing relationships on public networks,” *First Monday*, vol. 2, no. 9, 1997.

-
- [8] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.
- [9] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Bguelin, “Formal verification of smart contracts: Short paper,” in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*. ACM, 2016, pp. 91–96.
- [10] B. Egelund-Müller, M. Elsmann, F. Henglein, and O. Ross, “Automated execution of financial contracts on blockchains,” *Business & Information Systems Engineering*, vol. 59, no. 6, pp. 457–467, 2017.
- [11] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, “Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 79–94.
- [12] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, “Demystifying incentives in the consensus computer,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 706–719.
- [13] “EtherDice smart contract,” <http://etherdice.io/>, aug 2015, accessed on April 15, 2018.
- [14] J. C. Hull, *Options, Futures and Other Derivatives, 9th edition*. Prentice Hall, 2014.

- [15] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 254–269. [Online]. Available: <http://doi.acm.org/10.1145/2976749.2978309>
- [16] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on ethereum smart contracts (sok),” in *International Conference on Principles of Security and Trust*. Springer, 2017, pp. 164–186.
- [17] D. Shane, “A crypto exchange may have lost \$145 million after its CEO suddenly died,” <https://www.cnn.com/2019/02/05/tech/quadriga-gerald-cotten-cryptocurrency/index.html>, February 2019, online: Accessed; 2019-03-05.
- [18] Hyperledger, “Five hyperledger blockchain projects now in production,” <https://www.hyperledger.org/blog/2018/11/30/six-hyperledger-blockchain-projects-now-in-production>, November 2018, online: Accessed; 2019-03-06.
- [19] N. Meiremans, “Hyperledger projects in real companies,” <https://medium.com/coinmonks/hyperledger-projects-in-real-companies-35016745362c>, July 2018, online: Accessed; 2019-03-07.
- [20] “Healthcare rallies for blockchains: Keeping patients at the center,” <https://www.ibm.com/downloads/cas/BBRQK3WY>, online: Accessed; 2019-03-07.
- [21] “White Paper: Blockchained Cannabis DNA: THE INFOR-

- MATION CHAIN FOR ADVANCED GROWERS AND REGULATORS,” https://cdn2.hubspot.net/hubfs/3402974/Medicinal%20Genomics%20Blockchained%20Cannabis%20DNA.pdf?t=1523386278442&utm_campaign=Validation&utm_source=hs_automation&utm_medium=email&utm_content=57554917&_hsenc=p2ANqtz-_0ukN97Yf_qJtgWnKrkKd8eq7WE5cB7PRwFkfDZqJyNkvtN6HyjpQLrq2EdEYdxA5T8-DybGIGtJiTdzpENUgU&_hsmi=57554917, Online: Accessed; February 06, 2019.
- [22] N. Dominguez, “Use Cases for Ethereum,” <https://coindiligent.com/ethereum-real-world-use>, January 2019, Online: Accessed; 2019-03-07.
- [23] G. Sharma, “London Stock Exchange and IBM to pilot blockchain for European SMEs,” <https://www.ibtimes.co.uk/london-stock-exchange-ibm-pilot-blockchain-european-smes-1631014>, July 2017, Online: Accessed; 2019-03-07.
- [24] A. Kharpal, “Barclays used blockchain tech to trade derivatives,” <https://www.cnbc.com/2016/04/19/barclays-used-blockchain-tech-to-trade-derivatives.html>, April 2016, Online: Accessed; 2019-03-07.
- [25] “Nasdaq Linq Enables First-Ever Private Securities Issuance Documented With Blockchain Technology,” <http://ir.nasdaq.com/news-releases/news-release-details/nasdaq-linq-enables-first-ever-private-securities-issuance>, December 2015, Online: Accessed; 2019-03-07.
- [26] P. Rizzo, “Chiles Largest Stock Exchange Plans to Implement IBM Blockchain Tech,” <https://www.coindesk.com/>

- [chiles-largest-stock-exchange-plans-implement-ibm-blockchain-tech](#), May 2017, Online: Accessed; 2019-03-07.
- [27] M. del Castillo, “TMX Selects Hyperledger For Blockchain Voting Prototype,” <https://www.coindesk.com/tmx-selects-hyperledger-blockchain-voting-prototype>, April 2017, Online: Accessed; 2019-03-07.
- [28] M. D. Castillo, “UBS Unveils Blockchain for Trade Finance at Sibos,” <https://www.coindesk.com/ubs-blockchain-prototype-trade>, September 2016, Online: Accessed; 2019-03-07.
- [29] A. Back, “Hashcash-a denial of service counter-measure,” 2002. [Online]. Available: <http://www.hashcash.org/papers/hashcash.pdf>
- [30] S. King, “Primecoin: Cryptocurrency with prime number proof-of-work,” *July 7th*, 2013.
- [31] S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake,” *self-published paper, August*, vol. 19, 2012.
- [32] P. Vasin, “Blackcoins proof-of-stake protocol v2,” *URL: https://blackcoin.co/blackcoin-pos-protocolv2-whitepaper.pdf*, 2014.
- [33] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, “proof of activity: Extending bitcoin’s proof of work via proof of stake [extended abstract] y,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 3, pp. 34–37, 2014.

- [34] D. Schwartz, N. Youngs, A. Britto *et al.*, “The ripple protocol consensus algorithm,” *Ripple Labs Inc White Paper*, vol. 5, 2014.
- [35] B. Chase and E. MacBrough, “Analysis of the xrp ledger consensus protocol,” *arXiv preprint arXiv:1802.07242*, 2018.
- [36] “Proof of Stake FAQs,” <https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs>, online: Accessed; February 06, 2019.
- [37] V. Buterin and V. Griffith, “Casper the friendly finality gadget,” *arXiv preprint arXiv:1710.09437*, 2017.
- [38] J. Ray, “Casper Proof of Stake compendium,” <https://github.com/ethereum/wiki/wiki/Casper-Proof-of-Stake-compendium>, online: Accessed; February 11, 2019.
- [39] V. Zamfir, “Casper the Friendly Ghost: A Correct-by-Construction Blockchain Consensus Protocol,” <https://github.com/ethereum/research/blob/master/papers/CasperTFG/CasperTFG.pdf>, online: Accessed; February 11, 2019.
- [40] “Schneider J, Blostein A, Lee B, Kent S, Groer I, Beardsley E (2016) Blockchain-putting theory into practice.” <http://www.finyear.com/attachment/690548/>, online: Accessed; 2019-02-07.
- [41] J. P. Moyano and O. Ross, “KYC optimization using distributed ledger technology,” *Business & Information Systems Engineering*, vol. 59, no. 6, pp. 411–423, 2017.

-
- [42] P. Bahr, J. Berthold, and M. Elsmann, “Certified symbolic management of financial multi-party contracts,” *ACM SIGPLAN Notices*, vol. 50, no. 9, pp. 315–327, 2015.
- [43] I. Bentov, A. Mizrahi, and M. Rosenfeld, “Decentralized prediction market without arbiters,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 199–217.
- [44] M. Rosenfeld, “Overview of colored coins,” <https://bitcoil.co.il/BitcoinX.pdf>, 2012, online: Accessed; 2018-07-05.
- [45] “Oraclize,” <http://docs.oraclize.it/>, online: Accessed; 2018-04-18.
- [46] “Buterin Vitalik (2014), SchellingCoin: A Minimal-Trust Universal Data Feed,” <https://goo.gl/w2aJwu>., online: Accessed; 2018-04-18.
- [47] S. Eskandari, J. Clark, V. Sundaresan, and M. Adham, “On the feasibility of decentralized derivatives markets,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 553–567.
- [48] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, “Astraea: A decentralized blockchain oracle,” *arXiv preprint arXiv:1808.00528*, 2018.
- [49] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, “Zeus: Analyzing safety of smart contracts,” in *25th Annual Network and Distributed System Security Symposium (NDSS18)*, 2018.
- [50] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bueznli, and M. Vechev, “Securify: Practical security analysis of smart contracts,” in *Proceedings of the*

- 2018 ACM SIGSAC Conference on Computer and Communications Security*.
ACM, 2018, pp. 67–82.
- [51] S. Amani, M. Bégel, M. Bortin, and M. Staples, “Towards verifying ethereum smart contract bytecode in isabelle/hol,” in *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, 2018, pp. 66–77.
- [52] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: a proof assistant for higher-order logic*. Springer Science & Business Media, 2002, vol. 2283.
- [53] “A blockchain platform for the enterprise, hyperledger fabric,” <https://hyperledger-fabric.readthedocs.io/en/latest/>, online Accessed : February 8, 2019.
- [54] “Quorum Whitepaper,” <https://github.com/jpmorganchase/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf>, online: Accessed; February 08, 2019.
- [55] “Go Ethereum,” <https://github.com/ethereum/go-ethereum>, online: Accessed; February 08, 2019.
- [56] “The Corda Platform: Blockchain for every business in every industry,” <https://www.r3.com/corda-platform/>, online: Accessed; February 08, 2019.
- [57] Richard Gendal Brown, James Carlyle, Ian Grigg, Mike Hearn, “Corda: An Introduction,” https://docs.corda.net/_static/corda-introductory-whitepaper.pdf, online: Accessed; February 08, 2019.

-
- [58] “What is IOTA?” <https://www.iota.org/get-started/what-is-iota>, online: Accessed; February 08, 2019.
- [59] “DTCC Selects IBM, AXONI and R3 to Develop DTCC’s Distributed Ledger Solution for Derivatives Processing,” <http://www.dtcc.com/news/2017/january/09/dtcc-selects-ibm-axoni-and-r3-to-develop-dtccs-distributed-ledger-solution>, online: Accessed; February 08, 2019.
- [60] “Pricegeth,” <https://github.com/VelocityMarket/pricegeth>, online: Accessed; 2018-04-18.
- [61] “On the feasibility of decentralized derivatives markets,” <http://ongxabeou.over-blog.com/2017/12/on-the-feasibility-of-decentralized-derivatives-markets.html>, online: Accessed; 2018-05-28.
- [62] “Solidity,” <https://solidity.readthedocs.io/en/v0.4.22/>, online: Accessed; 2018-04-18.
- [63] “Remix - solidity ide,” <http://remix.readthedocs.io/en/latest/>, online: Accessed; 2018-04-18.
- [64] “Debugging a dapp using remix - mist - geth,” https://remix.readthedocs.io/en/latest/tutorial_mist.html, online: Accessed; 2019-03-13.
- [65] “The DAO Attack,” <https://www.coindesk.com/understanding-dao-hack-journalists/>, may 2016, accessed on April 15, 2018.