TECHNISCHE
UNIVERSITÄT
DRESDEN

Faculty of Computer Science Institute of Software and Multimedia Technology

Chair of Software Technology

Bachelor Thesis

Contract to Contract Calls for Financial Applications in Algorand

Vasil Petrov

Born on: April 8, 1999 in Burgas, Bulgaria
Matriculation number: 4818111
Matriculation year: 2018

to achieve the academic degree

Bachelor of Science (B.Sc.)

Supervising professor
Prof. Dr. Uwe Aßmann

Submitted on: August 17, 2022

Contents

Abstract

here goes the abstract, I am going to write it when I am done with the rest, see you then ;)

1 Introduction

1.1 Motivation

Blockchain technology has the potential to revolutionize the financial world. Its industrial adoption is growing at a rapid rate mainly due to the rising popularity of cryptocurrency. Big names in the banking sector such as Bank of America, Agricultural Bank of China, HSBC, BNP Paribas and many more have already invested a lot of capital into researching and experimenting with blockchain solutions for their financial services with the hope of reducing processing costs and time.[15]

In simple terms, blockchain represents a decentralized immutable ledger where transactions and assets can be securely stored and processed. Through cryptography and other mechanisms used by this technology, the need for a trusted central authority as a middleman in the transaction between two individuals is redundant. This could open the doors to the global financial market for a lot of people that don't have access to modern banking services.[17] Fintech startups are quickly trying to capture this new market by developing financial applications on various blockchain platforms like Algorand for example. Algorand is a cutting-edge blockchain technology created by Silvio Micali, a professor at the Massachusetts Institute of Technology and a Turing Award winner, and his team of leading scientists. In this thesis, we will look at the qualities and possibilities of developing financial applications specifically on Algorand.

Smart contracts play a key role in the development of blockchain-based applications. They can be interpreted as programs that are stored on the distributed ledger and execute some logic automatically when certain conditions are met. As suggested in this paper[18] by T. Mattis and R. Hirschfeld, an analogy could be made between deployed smart contracts and objects in object-oriented programming. Each contract has its own *states* and *methods* and plays its specific part in the application.

Object-oriented programming (OOP) is a paradigm that is used for designing modular, reusable software. With this approach, a complex problem could be solved by dividing it into a set of subproblems among different classes, making the development process easier and more intuitive. In the blockchain world, this could be achieved using contract composability which refers to combining multiple smart contracts, where each solves a certain aspect of the global problem, achieving again a modular structure. The way to achieve composability is by contract to contract calling. This allows one smart contract to interact and execute transactions to another contract on its own. Thus, with just one transaction sent by the client of the application, a chain of complex logic and communication between different contracts can be triggered, performing the requested task. This can serve as an example that despite the new ways of solving existing problems through blockchain, the well-established principles and practices for software development and design remain the same.

An important feature of blockchain is that once something is stored on the ledger, it becomes immutable. However, this can sometimes be bad for software development, because often certain parts of the program need to be optimized or rewritten. Luckily there are ways to get around this drawback by using upgradable smart contracts. The idea is that all transactions and values that have already happened on-chain will be visible and immutable on the ledger until it exists. However, we can still swap the logic of a smart contract with a new upgraded one. In this thesis, we will examine how this can be achieved on Algorand and other famous blockchain protocols.

Combining the OOP programming approach with blockchain could be a very powerful concept. However, there still exists a gap in research and development into blockchain for financial applications from an academic perspective, and this gap could hurt the adoption of this relatively new technology by the banking sector. For this reason, the main focal point of this thesis will address the topic of role-based programming in the context of blockchain development using Algorand. As mentioned in [23], the idea behind role-oriented programming is that the same object (entity) in different contexts could play different roles. Each role serves a specific purpose and can overwrite the behavior of the object allowing it to adapt to a certain context dynamically. For example, a customer of a bank can be at the same time a borrower and depositor. There is no point to model borrower and depositor as separate objects, but instead, as roles that can be added or dropped dynamically by customer objects. This way we can achieve a more accurate and natural representation of the real world in our program.

The primary task of this thesis will be to answer the following research questions:

1. How to make the Algorand blockchain interact and be a part of a role-based application?
2. How features such as contract to contract calling and upgradable smart contracts can help the Algorand blockchain to implement a role-oriented approach?
3. What are the drawbacks and advantages of developing role-based applications using the programming language of Algorand?

1.2 Goal and objectives

To answer the research questions asked in Motivation, a simple banking app will be created. The Algorand blockchain platform is going to be used for the development of the program together with python as a general-purpose language. In its implementation, I will demonstrate the relationship between OOP objects and smart contracts. Roles will also be presented by a client and an investor class that are going to communicate with a smart contract representing the bank. Depending on the role, different functionalities will be accessible to the object. Services such as opening a bank account, depositing, withdrawing and transferring funds will also be used to demonstrate contract to contract calling features. Contract composability and upgradeability will be applied to show how smart contracts could communicate and cooperate to build a more complex application.

1.3 Structure of the thesis

This thesis is divided into seven chapters. First and foremost, the topic of the research is laid out together with the goals and objectives in the Introduction section. The reader should be familiar with some basic concepts to better grasp and follow the research process, this is done in the Background chapter. Concepts like role-based programming, blockchain, smart contracts and the Algorand protocol will be explained in more detail.

Then comes Problem analysis, dedicated to the description of the bank application to be created. The structure of the app will be presented in detail with the help of Class and Sequence UML diagrams, explaining each important functionality. It will also be answered how this app should solve the research questions. The tech stack used for its implementation will also be mentioned.

In section four comes the practical part. A detailed description supported by code snippets will illustrate the development process and how the bank application part by part is being created.

The test results regarding the functionality of the developed app are shown in the Evaluation section of this thesis. An argument will be presented on how these results provide an answer to the research questions.

In the Discussion chapter, the pros and cons of developing applications on the Algorand blockchain will be considered. The arguments will also be supported by performance tests. Reference and comparison to another popular blockchain platform, namely Ethereum, will also be made here.

Finally, the Conclusion section presents perspectives for the extension of the presented research and app as well as a summary of the researched questions and process.

2 Background

In this chapter, the necessary basic knowledge of different concepts is provided which is key for understanding the rest of the thesis. We begin with a more in-depth definition of blockchain technology and what advantages it offers. After that, smart contracts are going to be discussed with an explanation of features such as contract to contract calling and upgradeable contracts. Then we will continue with a description of the Algorand blockchain and its perks. Finally, we will end with a more detailed explanation of the role-based programming approach.

2.1 Blockchain technology

Many consider blockchain as a revolutionary technology that is the next big thing. It is even compared in the potential to the early years of the internet. However, many do not know that its basic idea and fundamentals predate the emergence of cryptocurrencies.

2.1.1 History and Structure

David Chaum in his dissertation from 1982 called "Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups"[8] is one of the first to suggest a protocol that resembles blockchain as we know it today. He proposed the implementation of a decentralized system that can be maintained and trusted by achieving mutual consensus between its participants. Almost a decade later S. Haber and W. Scott Stornetta proposed in their work[14] a time-stamping mechanism for digital documents. The process involves a one-way hash function where the hashes of the different time-stamped documents are linked together, forming a tamper-proof cryptographically secured chain. Based on these and some other early works, Satoshi Nakamoto published a whitepaper titled "Bitcoin: A Peer-to-Peer Electronic Cash System"[20], where blockchain was introduced as a public distributed ledger for bitcoin payment transactions. From this point on, the rising popularity of this technology and cryptocurrencies began.

As explained by A. Panwar and V. Bhatnagar in [21], blockchain is just a type of distributed ledger technology (DLT). This is important to note because many people use both terms interchangeably which is technically wrong. DLT is a decentralized database distributed as copies across multiple computer nodes connected via a peer-to-peer (P2P) network. If a modification occurs, then all copies of the ledger get autonomously updated and synchronized. With the things said so far, it sounds exactly like the definition of blockchain. However, the key difference is that DLT doesn't necessarily need to represent its data in a block

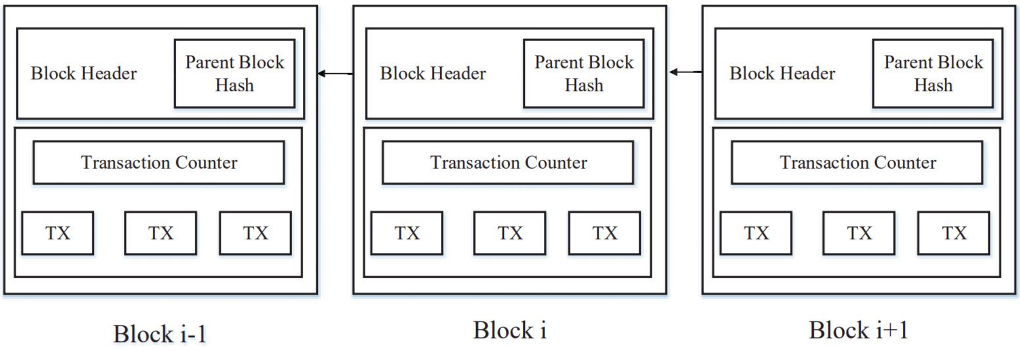

Figure 2.1: An example of a basic blockchain structure (Figure 1 in [30])

structure, but in any type of structure. Furthermore, DLT doesn't necessarily store data in a specific sequence. Many blockchains also use some kind of native currency/token on the network, used to pay fees for interacting with the network and also as an exchange value between different participants. It is also given as a reward for the people who keep the blockchain alive and add/validate new blocks to the chain. However, not all forms of DLT require a token or a currency to keep the system going. These are just some of the key differences between the two technologies.

As described by Z. Zheng et al.[30], blockchain can be viewed as a sequence of blocks that are linked together in a chronological order using cryptography. As can be seen on Figure 2.1, each block is composed of a *block header* and a *block body*. The block header holds the hash value that points to the previous block (also called a parent block) and a timestamp that shows when the block was validated and added to the chain. In each block body, a certain number of transactions is being stored depending on the size of the block and each transaction. Any attempt to modify some information in an existing block will affect its hash address and all others before and after it because everything is connected. The participants will compare the change with their current copy of the ledger and immediately notice the change and reject it. This is why blockchain is considered immutable and tamper-proof.

As further mentioned in [30], asymmetric cryptography (public-key cryptography) is one of the key components of blockchain technology. Each participant in the network owns a pair composed of a private key and a public key. The private key is used to verify transactions and prove ownership of a blockchain address and the public key is used for the unique account address, allowing users to receive cryptocurrencies or other digital assets.

2.1.2 Types of blockchain

Blockchain networks can be divided into 3 categories: public, private and consortium.[16]

Public blockchains are permissionless, meaning that they are open networks, where anyone can participate and read/write to the ledger without needing the approval of any trusted authority. The transactions are completely transparent and accessible for anyone to see, but in most cases they are anonymous and you can't directly link them to the person behind them. Here we see anonymity and transparency, qualities for which blockchain is liked and which are absent in the standard financial system.

Private blockchains on the other hand require permission to join them. They are more suitable for organizations and companies which want to benefit from its implementation but don't want everyone to have control over the network. Normally, it is again a centralized structure, where only some participants have the right to verify and add new data to the chain.

In consortium blockchains, there is more than one company/organization in charge to decide who has what rights on the blockchain. It is best suited for organizational collaboration. It can be viewed as a hybrid between public and private blockchain because the power is not only in one central structure, but at the same time, not everyone can participate and interact with the network.

From this moment forth, the word blockchain will refer to the public version of the network, because this thesis will revolve around Algorand, which is a permissionless public blockchain.

2.1.3 Consensus mechanisms

Based on the provided knowledge so far, blockchain is a decentralized system available for everyone to take part in it. Everyone has an updated and synchronized copy of the ledger. The control of the network is not in the hands of some central entity but all participants in the network. However, what guarantees that the copy of each node will be up to date and correct? Who validates and adds new transactions/data to the chain? How is it verified that these new transactions are valid at all? Here comes the important role of a consensus mechanism. The most used ones are Proof of Work (PoW) and Proof of Stake (PoS), which are going to be briefly explained in the following paragraphs.

PoW was the first consensus mechanism ever for blockchain. It was introduced in the whitepaper for Bitcoin[20], where Nakamoto argued that this will be the mechanism to keep the distributed ledger secure and consistent while being decentralized and without a central authority to regulate. As explained in the research article "On the Security and Performance of Proof of Work Blockchains"[12] by A. Gervais et al., the PoW principle of selecting a node to add a new block to the chain is by propagating to the network a cryptographic puzzle that needs to be solved. Every node also called a miner, is in direct competition with the others and spends computational resources and time to find the solution. Once an answer is found, it gets propagated to the network together with the new block and the other nodes verify if they are valid or not. If yes, the new block gets appended to the chain and all participants update their ledgers. If not, the block will be rejected and a new proposal will be expected. This process is repeated and thus the chain is kept consistent, valid and secure.

PoS has a different approach to reaching a consensus. As described by F.Saleh in [22], PoS chooses the proposer (forger) of a new block and the validators based on their stake or in other words on the proportion of native coins they are holding. The protocol chooses a random coin each round from the supply and the node to whom it belongs will be entitled to propose a new block on the chain. This is one of the reasons why PoS is considered a more elegant mechanism than PoW and its use in new blockchain projects is increasing.[22] They are better in terms of scalability and require much fewer resources such as electricity or investment in equipment to participate.

2.1.4 Summary of blockchain's characteristics

As a summary of the previous three subsections, blockchain is an immutable, decentralized, distributed, transparent and secure network that is maintained and managed by all its participants that run its software. These participants are called nodes and each has its copy of the ledger that holds all of the transactions and information that are so far written on the chain. This supports the claim of distributed and transparent nature of the system. Furthermore, the authority and control belong to all the nodes, not to a single central structure. With the help of a consensus mechanism, the participants can collaborate without problems in adding new blocks to the chain and maintaining its integrity and consistency. This achieves decentralization and security of the network and a transaction is accepted only if the majority of nodes agreed that it is valid. Thanks to the cryptography used in the protocol, already existing records on the chain can't be edited or deleted. This guarantees the immutability of the transactions made on the blockchain.

2.2 Smart contracts

Companies from various fields are looking for ways to optimize the quality and efficiency of their services and products through blockchain. Every one of them has some arrangement with their customers. For instance, if you pay an X amount of money, you get some service Y in return. If you don't like the quality of the service or product, then you can complain and get your money back. This is a simple example, but any kind of business operates more or less on the same principle. To recreate this logic in blockchain, we need the concept of smart contracts.

2.2.1 History and meaning

The term *smart contract* was coined for the first time by American cryptographer Nick Szabo back in the 90s.[25] That was long before the Bitcoin whitepaper[20] and the emergence of blockchain as a technology. It was described by Szabo as a digital form of a contract between two or more parties that can give you automatically an outcome as soon as some predetermined conditions are met. The main idea of this tool was to automate and optimize the workflow of business processes and eliminate the role of intermediaries that act as trust between the parties. The thing that would monitor the proper execution of the terms of the agreements and build trust between the different parties is the code itself. This definition is still relevant today, but it took nearly two decades before blockchain put smart contracts into practice.

As explained in the Ethereum whitepaper[4] by Vitalik Buterin, the scripting language for the Bitcoin protocol has its boundaries. It has limited programmability for the sake of reducing potential vulnerabilities and increasing the security of the decentralized currency. For instance, loops are not allowed in order to evade infinite loops. This makes the scripting language of Bitcoin non-Turing complete. Another problem mentioned by Buterin is the lack of state, meaning that the Bitcoin blockchain can't distinguish other states of a transaction except for *spent* and *unspent*, making it impossible to implement multi-stage smart contracts, which require more options and intermediate states to replicate real-world conditions and agreements. In other words, even after the creation of Bitcoin and the emergence of blockchain as a protocol, smart contracts still couldn't be implemented. A few more years had to pass to turn the cooperation between the two concepts into reality.

According to [27], Ethereum was the first blockchain to enable smart contract programming and remains the most widespread platform for developing decentralized applications. This is done via the Ethereum virtual machine (EVM) which is Turing-complete. Now smart contracts can exist as programs on the blockchain, read/write values on it, execute transactions, exchange assets or call and collaborate with other contracts to create complex application logic running on the ledger. Because of the distributed nature and immutability of the blockchain, it could be guaranteed that no one can edit or delete an already deployed smart contract and that if the specific conditions are met, a certain outcome will happen. Furthermore, each contract can be represented as a regular account that can hold its own assets/funds and has its own unique address on the network. Since the appearance of Ethereum, smart contracts have been an essential component of blockchain technology and many companies are taking advantage of their qualities. Algorand also uses smart contracts and allows the development of applications on its platform, but more details on this are provided in the Algorand subsection of this chapter.

2.2.2 Upgradable smart contracts

As stated here [1], once a smart contract is deployed on the blockchain, it becomes immutable. If a new condition needs to be added or a security issue to be fixed to the existing code, then a new version of the contract should be deployed. This is not at all convenient for application development because it requires very careful pre-consideration and is also expensive to maintain. If you swap a smart contract with a newer version, then you have to inform all other contracts and users that interact with it about its new address and functionality. Otherwise, the old version will continue to be called and the update will be in vain. This drawback could prove to be a big barrier for companies looking to adopt blockchain solutions to their business. Luckily, people from academia and industry have come up with some ways around the immutability problem.

As mentioned in "Evaluating Upgradable Smart Contract"[3] by Van Cuong Bui et al., smart contracts on Ethereum can't be modified once deployed, but by using design patterns we can work around the code's upgradeability issue. Each contract consists of logic (functions) and state (data). The proxy pattern uses the idea of splitting the smart contract into one responsible for the data storage and the other for the logic. As shown in Figure 2.2 the proxy contract is the one taking care of the data and all the storage changes are happening there. The proxy also stores in its state the address of the current logic contract. As described in [3], when a caller (user) calls the proxy, it delegates the call to the current logic contract where all of the functionality is. Because we have delegation, the functions being executed in the logic contract will be in the context of the proxy, meaning they will use the proxy's data storage. Then the logic contract will return a result to the original caller again via the proxy. As highlighted in the figure below, if we want to add some new functionality, we just deploy a new logic contract and tell the proxy its address. From this moment forward, the proxy will delegate all external calls to the new version. With the proxy pattern upgradability of smart contracts is achieved without hurting the immutability factor. No contract gets deleted or modified, just the logic contract is swapped with a new one and the proxy starts to delegate calls from users or other smart contracts to the newer version instead of the older one. This is also a lot easier to maintain because now only the proxy needs to be informed of the new address.

It is important to remark that on Algorand these kind of patterns are most of the time not required. Contrary to Ethereum, smart contracts can be updated directly, and the right to update is controlled by the application itself. However, only the logic part of a contract can be updated without the need to deploy a new version. The amount of data storage remains immutable. In the end, we achieve the same result as by using design patterns. More on the matter will be explained in the Algorand subsection of this chapter and Implementation.

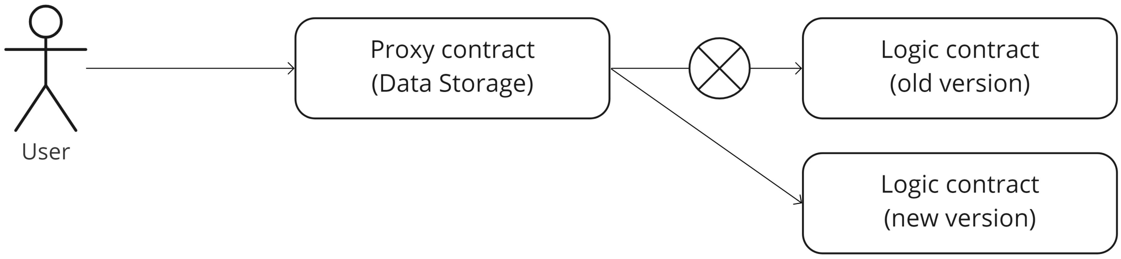

Figure 2.2: Proxy Pattern

2.2.3 Contract to contract calls

For blockchain to be adopted by large companies, then writing complex applications on it must be possible. We have found that with smart contracts this can be achieved. Nonetheless, we cannot afford to write all the business logic in just one contract. First, it is against the good practices of software engineering. If we want the code to be easier to maintain, we need modularity and interoperability between the different building blocks of the program. Second, smart contracts are also limited in their size. On Ethereum for example, the maximum size of a contract is limited to 24KB [1], on Algorand it is 8KB [2]. These limitations further reduce the possibility of writing large programs in just one contract.

This is why the possibility of communication between smart contracts at runtime is crucial for developing complex applications that follow good practices. As described in this article[29][3], Algorand allows a smart contract to call, share information and even create other smart contracts. This enables developers to build powerful and versatile decentralized applications on the Algorand blockchain.

2.3 Algorand

2.3.1 Goals and advantages

Algorand is one of the most advanced open source permissionless public blockchains currently in the field. Founded in 2017 by MIT professor and Turing Award winner Silvio Micali, the goal of the Algorand protocol is to solve the blockchain trilemma or in other words, to be a truly decentralized, secure, and scalable system.[10] Micali and his colleagues saw that most blockchains like Bitcoin or Ethereum have weaknesses in fulfilling these 3 requirements simultaneously because of their consensus mechanism PoW and because of their overall architecture.[9] As noted in [11], some of the flaws of using PoW are that it requires too many resources and time to reach consensus, short-lived forks are a possibility and there is some degree of centralization because of the big mining pools, where miners combine their computational efforts to increase chances of proposing new blocks and get rewards for it. As explained in [26], forks happen when the blockchain splits into alternative paths, often happening because of disagreement between miners about which block should be the next added to the chain. Eventually, all of the miners go back to the longest version of the chain and the alternative ones get abandoned. As mentioned in [13], this is why users of the Bitcoin blockchain have to wait up to an hour to be certain that their transactions were confirmed and added to the main chain and not on a forked version that will be discredited further down the line. Forks are the cause of uncertainty and delay on the blockchain network and should be somehow avoided.

This is the reason why Algorand was created to solve most of these issues and accelerate the development of blockchain technology and its mass adoption. This is achieved by relying on algorithmic randomness for reaching a consensus between the nodes, which implies also its name - Algorand.[9] It combines PoS concepts with the Byzantine agreement protocol creating the unique consensus algorithm for Algorand called Pure Proof of Stake (PPoS).[9, 10] With it, the Algorand protocol achieves 125 times Bitcoin's throughput by staying at the same time secure and requiring insignificant resources to be operational.[13, 11] Instant transaction finality is also guaranteed because of the extremely low probability of the chain being forked. Thanks to this, transactions can be validated and accepted on the chain for less

[1] https://ethereum.org/en/developers/docs/smart-contracts/#limitations

[2] https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/#creating-the-smart-contract

[3] https://developer.algorand.org/articles/contract-to-contract-calls-and-an-abi-come-to-algorand/

than a minute and the latency remains constant even when scaled up with more network participants.[13]

2.3.2 Pure Proof of Stake

As mentioned in [13], Algorand's security depends on the belief that the majority of the stake is in non-malicious hands. If more than 2/3 of the ALGO supply (Algorand's native currency) is owned by honest participants, then consensus can be achieved while tolerating malicious users and their attacks. This is possible because nodes are randomly and secretly selected to take part in the proposal and voting process. As long as an account has at least one ALGO in its possession it can be selected to participate in the consensus. The probability for this to happen is directly proportionate to the number of ALGO a node holds.

As can be observed in Figure 2.3, the process of reaching a consensus and appending a new block to the chain can be divided into 3 phases: selecting nodes for block proposals, filtering the proposals down to just one, and voting whether it should be added to the chain or rejected. Before the beginning of each phase, the nodes engage in something called cryptographic sortition to find out whether they are selected to participate or not. [10] The whole consensus process goes like this:

- Block proposal - For each ALGO that a node holds, a Verifiable Random Function[19] (VRF) is being computed. All nodes do this in parallel, secretly and independently. VRF is a cryptographic function that requires negligible time and resources and at the end outputs a hash value that will determine whether the node was chosen to propose a block or not. There will be also a short VRF proof that is easily verifiable from others that the proposal is truly valid and rightful.[10, 2] If the node was chosen, it combines some pending transactions into a new block and propagates it to the network using the gossip protocol (the protocol that nodes use for communication on the Algorand blockchain).[13, 10]

- Filtering down to a single block - Again all nodes run the VRF as part of the cryptographic sortition to establish a committee. For a specific time period, each of the selected nodes will receive at least a few proposals and its job is to check the VRF proof and compare which of all the blocks has the highest priority. In the end, the committee has to select only one block, which goes to the final phase.[10, 2]

- Block certification - A new committee is created again with the sortition algorithm. This time each ALGO that has a "winning" VRF hash will equate to one vote for this node. In other words, a node can vote a couple of times whether the new block should be added to the chain or not. The block is checked for things like overspending, double-spending, and any other flaws. If it is okay and the block gets a certain number of votes that are more than a certain threshold, then the block gets appended to the chain. This is the end of the round and in the next round, the whole process repeats itself. [13, 10, 2]

The reason why this consensus can operate securely even when there are some malicious nodes in the network is because of the cryptographic sortition. Each phase of the consensus process involves a different group of nodes, where each node is privately chosen to participate. In this way, an adversary doesn't know which node to corrupt or attack. By the time it is clear which nodes are part of the committee, it is already too late because their proposals or votes are already propagating through the network.[13, 2] It is important to note that all of these steps are happening within seconds, but even if the adversary is powerful and quick enough to execute attacks, it just won't know who to target. By virtue of its consensus mechanism, Algorand achieves speed, efficiency, security, and true decentralization.

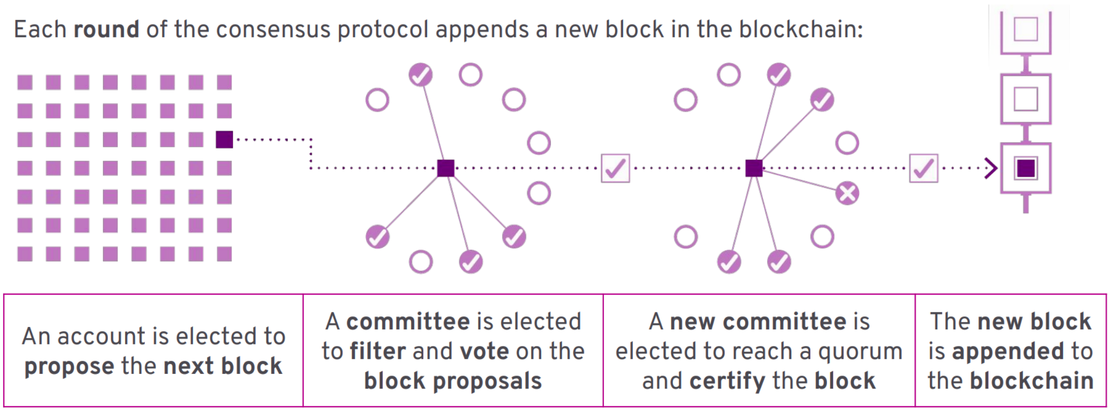
Each round of the consensus protocol appends a new block in the blockchain:
An account is elected to propose the next block
A committee is elected to filter and vote on the block proposals
A new committee is elected to reach a quorum and certify the block
The new block is appended to the blockchain

Figure 2.3: Pure Proof of Stake (Taken from [2])

2.3.3 Algorand smart contracts

Algorand is a programmable blockchain just like Ethereum. This means that smart contracts can also be written and evaluated on it. According to [7], in comparison to Ethereum, Algorand is more efficient and the execution of smart contracts happens for significantly less time and lower fees. There are two types of smart contracts for the Algorand blockchain - *stateful* and *stateless*.[6]

Stateful contracts live on the blockchain and hold some data (state) connected to it on the ledger. They can manipulate data by reading and writing it on the chain, execute and create transactions and perform logic operations as part of an application.[7, 6] Every such contract has an unique app ID and also an account that can hold ALGOs.

As mentioned in [28, 7] Stateless smart contracts (also known as Smart Signatures or Logic Signatures) are used for approving asset transactions between accounts. On Algorand, a transaction can be cryptographically signed either with the private key of the sender's account, a multi-signature (group of private keys of different accounts), or with a Logic Signature. These contracts aren't deployed to the blockchain as separate entities with their state, but rather attached to a transaction. The transaction will be successfully executed and accepted on the ledger only if the Logic Signature evaluates to true. Stateless smart contracts can be used also as escrow accounts between different parties by holding their funds and releasing them after some pre-agreed condition was met. Then the contract gets an address on the blockchain where the funds will be stored.[28, 7] In contrast to stateful contracts, they can't write/read state during evaluation.

The Algorand language used for writing and evaluating smart contracts and transactions on the blockchain is called Transaction Execution Approval Language (TEAL). TEAL is a low-level language with assembly-like syntax. Once the TEAL program is written and executed, it gets compiled into bytecode that is run on the Algorand Virtual Machine (AVM). The AVM is a Turing-complete execution environment on Algorand that works like a stack interpreter that processes TEAL programs and checks whether they are valid or not. If the program is valid, then the transaction will be sent to the network and the AVM will return a single non-zero value on top of the stack, signaling that the transaction was accepted.[7, 2]

Regarding the upgradability of the smart contracts[4], on Algorand they can be easily updated via a special type of transaction. If the transaction sender who wishes to update the contract is allowed to do so, then the new programs should be provided together with the app ID of the contract. However, this will update only the contract's logic, the state associ-

[4] `https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/#update-smart-contract`

ated with it will remain immutable and can't be changed once the contract is deployed for the first time. This is a lot more convenient in comparison to other blockchains that require some kind of design pattern to achieve smart contract upgradability.

Contract to contract calling is also possible on the Algorand blockchain since the introduction of AVM 1.1 (TEAL version 6).[5] This can be done via inner transactions.[29] This allows developers to build complex applications on the Algorand network that can include several smart contracts collaborating. In the Implementation chapter of this thesis, all of these concepts like Stateful smart contracts, upgradability and contract-to-contract calling will be demonstrated in a real application.

2.4 Role-based programming

In today's society, we as humans tend to see ourselves and others playing distinct roles depending on different situations. Let's take for example a person named Bob. At work, Bob is an employee. At home, he is a husband and a father. When he is buying some groceries, then he is a customer of a store. These are all different roles that Bob plays according to the context, but all of the time this is the same person called Bob. The role-oriented approach tries to module this concept and view of our world into the process of developing software.

As noted in "Refactoring to Role Objects"[24] by F. Steinmann and F. Stolz, we can use the concept of Inheritance in OOP to model the example with the person Bob and his roles. We can make Bob the superclass and all of his roles of employee, husband, etc., could inherit from it as subclasses. Because each subclass can be extended with certain unique functionality needed for its context and at the same time inherits the basic behaviors and specifications of the person Bob, it could be argued that with Inheritance we successfully achieved modeling Bob and his roles as a program. However, as correctly pointed out in [24], now each new role that we create for Bob can be interpreted as an independent object representing a different individual which is not reasonable compared to the real-world scenario where every role is represented by the same person called Bob.

To make an adequate representation of the given scenario with Bob, we can use the Role Object pattern (ROP). In Figure 2.4 taken from [5], we have a UML diagram of ROP.With the explanations provided in [5] and our example of Bob and his roles, the structures in the diagram will have the following meaning:

- **Component** - this is an abstract class that shows how role objects will be handled by the ComponentCore and also some general methods applicable for all ComponentCore instances. In our case, we can name this construct *PersonAbstract*, which is a general representation of people in society with their roles and how to manage them.

- **ComponentCore** - this is where a concrete instance of a person can be created. Bob will be an instance of this class. We can rename it to *Person*.

- **ComponentRole** - All concrete roles derive from this abstract class. It makes the connection with the ComponentCore by storing a reference to it (for example a reference to the object Bob).

- **ConcreteRole** -here all concrete roles of Bob can be implemented with their additional functionalities as separate classes like Employee, Customer, Husband, Father, etc.

As described in [24] and [5], using ROP gives us the advantage to alter the behavior of an object dynamically depending on the context with the help of adding or removing role

[5] `https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/#contract-to-contract-calls`

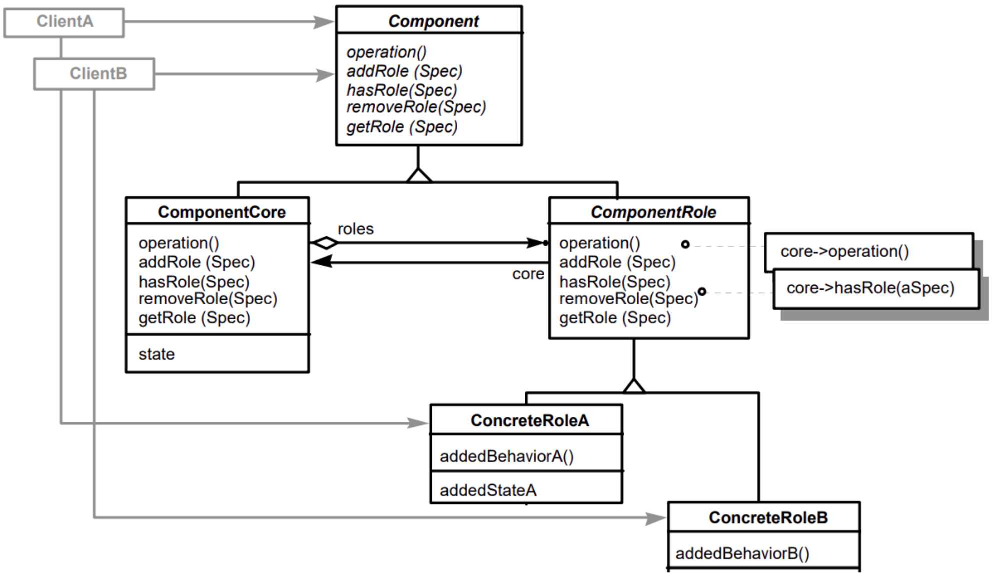

Figure 2.4: Structure diagram of the Role Object pattern (Figure 3 in [5])

objects at run time. In comparison to using Inheritance, the role objects will only extend the functionality of the core object and share its state, representing the same logical entity, namely the person Bob for instance. An object can hold many roles concurrently, each for its respective context. Each role can be also upgraded without the need to change anything in the core object or other roles. This means that with ROP we can achieve separation of contexts, low coupling, and easier code maintenance.

In summary, with role-based programming and Role Object pattern, we want to have objects that could have dynamically different attributes and behaviors according to specific contexts. This could be done when the object has different roles attached to it and can manage them at run time. Each role is independent and has its own purpose, but all of them collectively represent the same core object in different scenarios.

3 Problem analysis

The main objective of this thesis is to investigate whether a role-based application can collaborate with the Algorand blockchain. If possible, the development process should be documented and practically demonstrated with a Proof of Concept (PoC) targeting the financial sector.

A simple banking application can be planned that is modeled after a real-life bank where people can have different roles as clients or investors. Functionality such as opening a bank account, deposits, withdrawals, etc. should also be introduced. The key part is that all of the important data and transactions that take place in this bank should be recorded on the Algorand blockchain, where they will be securely stored and immutable. Furthermore, most of the bank functionality should be implemented on the blockchain as well via smart contracts. This will allow us to further examine and test the process of application development on a distributed ledger practically, not only theoretically. It will also help us understand whether a smart contract can react and execute different logic depending on the role that is calling it. All this will be done to bring forth the possibility that a real-world banking institution can replace its centralized way of storing data and doing operations with blockchain technology. Finally, after some testing, we can discuss what advantages and disadvantages this migration may bring.

3.1 SDKs and TEAL limitations

To develop a meaningful role-based banking application, we will most likely need to use OOP concepts such as Inheritance, Composition, Abstract classes, etc. Unfortunately, the language TEAL with which we can create applications on the Algorand blockchain is a low-level language with an assembly-like syntax.[1] This means that TEAL doesn't provide a high level of abstraction and doesn't have these OOP concepts built in. Due to this reason, the process of creating complex applications using just TEAL can be very inconvenient, which may discourage many companies from using Algorand for their services. Fortunately, Algorand provides some SDKs (software development kits) that make the whole development process more pleasant and easy.[2] There are SDKs for some of the most popular programming languages like Python, Java, JavaScript, C#, Rust, etc. with which you can interact with the Algorand network.

For the PoC that I am going to create for this thesis, I will be using the Python SDK and

[1] https://developer.algorand.org/docs/get-details/dapps/avm/teal/specification/

[2] https://developer.algorand.org/docs/sdks/

PyTeal[3]. PyTeal is a domain-specific language for Algorand that allows us to write code for smart contracts in Python that is going to be compiled to TEAL afterward.[2, 7] Now we have a more convenient syntax for creating smart contracts and also a higher level of abstraction that we can utilize. It is important to reiterate that PyTeal programs are not compiled directly into bytecode and passed to the AVM for processing but are first compiled into TEAL programs.[2] This ultimately means that PyTeal is still dependent on the TEAL limitations. We can't compile classes and objects written in Python to TEAL, but at least now we have a binding mechanism between Algorand and a high-level programming language thanks to the SDK. This indicates the possibility for a role-based application to tightly interact and collaborate with the Algorand blockchain and its smart contracts.

[3] PyTeal Documentation (2022) `https://pyteal.readthedocs.io/en/stable/index.html`

4 Implementation

4.1 Design of the PoC

5 Evaluation

6 Discussion

7 Conclusion

Bibliography

[1] Adnan, Imeri ; Lamont, Jonathan ; Agoulmine, Nazim ; Khadraoui, Djamel: Model of dynamic smart contract for permissioned blockchains. In: *2019 Practice of Enterprise Modelling Conference Forum (PoEM 2019 Forum)* Bd. 2586, 2019

[2] Bassi, Cosimo: *Algorand School*. 2022. – `https://github.com/cusma/algorand-school/blob/main/algorand-school-english.pdf`

[3] Bui, Van C. ; Wen, Sheng ; Yu, Jiangshan ; Xia, Xin ; Haghighi, Mohammad S. ; Xiang, Yang: Evaluating Upgradable Smart Contract. In: *2021 IEEE International Conference on Blockchain (Blockchain)*, 2021, S. 252–256

[4] Buterin, Vitalik: *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. 2014. – `https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf`

[5] Bäumer, Dirk ; Riehle, Dirk ; Siberski, Wolf ; Wulf, Martina: The Role Object Pattern. In: *Washington University Dept. of Computer Science*, 1998

[6] Chaudhury, Archie ; Haney, Brian: Algorand Autonomous. In: *SSRN* (2021). – https://ssrn.com/abstract=3819055

[7] Chaudhury, Archie ; Haney, Brian: Smart Contracts on Algorand. In: *SSRN* (2021). – https://ssrn.com/abstract=3887719

[8] Chaum, David: Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups, Ph.D. dissertation, Dept. Comput. Sci., Univ. California, Berkeley, 1982

[9] Chen, Jing ; Micali, Silvio: Algorand: A secure and efficient distributed ledger. In: *Theoretical Computer Science* 777 (2019), S. 155–183. – ISSN 0304–3975

[10] Conti, Mauro ; Gangwal, Ankit ; Todero, Michele: Blockchain Trilemma Solver Algorand Has Dilemma over Undecidable Messages. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*, Association for Computing Machinery, 2019 (ARES '19). – ISBN 9781450371643

[11] Fooladgar, Mehdi ; Manshaei, Mohammad H. ; Jadliwala, Murtuza ; Rahman, Mohammad A.: On Incentive Compatible Role-Based Reward Distribution in Algorand. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, S. 452–463

[12] Gervais, Arthur ; Karame, Ghassan O. ; Wüst, Karl ; Glykantzis, Vasileios ; Ritzdorf, Hubert ; Capkun, Srdjan: On the Security and Performance of Proof of Work Blockchains. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA : Association for Computing Machinery, 2016 (CCS '16). – ISBN 9781450341394, S. 3–16

[13] Gilad, Yossi ; Hemo, Rotem ; Micali, Silvio ; Vlachos, Georgios ; Zeldovich, Nickolai: Algorand: Scaling byzantine agreements for cryptocurrencies. In: *Proceedings of the 26th symposium on operating systems principles*, 2017, S. 51–68

[14] Haber, Stuart ; Stornetta, Wakefield: How to time-stamp a digital document. In: *Journal of Cryptology* 3 (1991), S. 99–111

[15] Hassani, Hossein ; Huang, Xu ; Silva, Emmanuel: Banking with blockchain-ed big data. In: *Journal of Management Analytics* 5 (2018), Nr. 4, S. 256–275

[16] Khan, Abdul G. ; Zahid, Amjad H. ; Hussain, Muzammil ; Farooq, M ; Riaz, Usama ; Alam, Talha M.: A journey of WEB and Blockchain towards the Industry 4.0: An Overview. In: *2019 International Conference on Innovative Computing (ICIC)*, 2019, S. 1–7

[17] Kshetri, Nir: Potential roles of blockchain in fighting poverty and reducing financial exclusion in the global south. In: *Journal of Global Information Technology Management* 20 (2017), Nr. 4, S. 201–204

[18] Mattis, Toni ; Hirschfeld, Robert: Activity Contexts: Improving Modularity in Blockchain-Based Smart Contracts Using Context-Oriented Programming, Association for Computing Machinery, 2018 (COP '18). – ISBN 9781450357227, S. 31–38

[19] Micali, S. ; Rabin, M. ; Vadhan, S.: Verifiable random functions. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, 1999, S. 120–130

[20] Nakamoto, Satoshi: Bitcoin: A Peer-to-Peer Electronic Cash System. (2008)

[21] Panwar, Arvind ; Bhatnagar, Vishal: Distributed Ledger Technology (DLT): The Beginning of a Technological Revolution for Blockchain. In: *2nd International Conference on Data, Engineering and Applications (IDEA)*, 2020, S. 1–5

[22] Saleh, Fahad: Blockchain without Waste: Proof-of-Stake. In: *The Review of Financial Studies* 34 (2020), Nr. 3, S. 1156–1190. – ISSN 0893–9454

[23] Steimann, Friedrich ; Stolz, Fabian U.: Refactoring to Role Objects. In: *Proceedings of the 33rd International Conference on Software Engineering*, Association for Computing Machinery, 2011 (ICSE '11). – ISBN 9781450304450, S. 441–450

[24] Steimann, Friedrich ; Stolz, Fabian U.: Refactoring to Role Objects. In: *Proceedings of the 33rd International Conference on Software Engineering*, Association for Computing Machinery, 2011. – ISBN 9781450304450, S. 441–450

[25] Szabo, Nick: Formalizing and Securing Relationships on Public Networks. In: *First Monday* 2 (1997), Nr. 9

[26] Vokerla, Rahul R. ; Shanmugam, Bharanidharan ; Azam, Sami ; Karim, Asif ; Boer, Friso D. ; Jonkman, Mirjam ; Faisal, Fahad: An Overview of Blockchain Applications and Attacks. In: *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, 2019, S. 1–6

[27] Wang, Shuai ; Ouyang, Liwei ; Yuan, Yong ; Ni, Xiaochun ; Han, Xuan ; Wang, Fei-Yue: Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49 (2019), Nr. 11, S. 2266–2277

[28] Weathersby, Jason: Linking Algorand Stateful and Stateless Smart Contracts. (2020). – `https://developer.algorand.org/articles/linking-algorand-stateful-and-stateless-smart-contracts/`

[29] Weathersby, Jason: *Contract to Contract calls and an ABI come to Algorand*. 2022, February 25. – `https://developer.algorand.org/articles/contract-to-contract-calls-and-an-abi-come-to-algorand/`

[30] Zheng, Zibin ; Xie, Shaoan ; Dai, Hongning ; Chen, Xiangping ; Wang, Huaimin: An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In: *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, S. 557–564

List of Figures

List of Tables

Statement of authorship

I hereby certify that I have authored this document entitled *Contract to Contract Calls for Financial Applications in Algorand* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, August 17, 2022

Vasil Petrov