Faculty of Media Engineering and Technology
German University in Cairo

Blockchain Security

Bachelor Thesis

Author: Abdallah Alkashawy
Supervisors: Dr.Gamal Ebrahim

Submission Date: 01 June, 2023

Faculty of Media Engineering and Technology
German University in Cairo

Blockchain Security

Bachelor Thesis

Author: Abdallah Alkashawy
Supervisors: Dr.Gamal Ebrahim

Submission Date: 01 June, 2023

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Abdallah Alkashawy
01 June, 2023

Acknowledgments

I would like to express my sincere gratitude to my supervisor Prof. Gamal Ebrahim,I would also like to express my gratitude to my parents who stood by me no matter the hardships.

Abstract

In this thesis, we delve into the exciting realm of web application development and explore how canisters, a vital component of blockchain technology, can revolutionize this domain with their decentralized approach. Traditional client-server models often face challenges related to scalability, security, and trust. However, canisters offer a promising alternative by leveraging the power of blockchain to create distributed and transparent applications.

Our journey commences with a deep dive into the architecture of blockchain, unraveling its decentralized nature. We also investigate the pervasive issue of double spending and investigate a range of innovative solutions that address this problem. Additionally, we explore the diverse applications of blockchain technology across industries, shedding light on its potential to climate, supply chain management, healthcare, and more.

Through meticulous research and practical analysis, we delve into the advantages and complexities of utilizing canisters for web application development. Our study encompasses a comprehensive examination of the fundamental principles and technical aspects underpinning canisters, including their architecture, programming model, and deployment procedures.

To demonstrate the potential of canisters, we embark on a captivating case study where we employ them to develop a fully functional web application. Throughout the development process, we tackle various challenges, such as data storage, user authentication, and transaction management, all within the dynamic canister environment. Importantly, we evaluate the performance of our application, highlighting the inherent advantages of canisters.

Our findings reveal that canisters provide an empowering and secure approach to web application development. By eliminating intermediaries and central points of control, canisters foster enhanced resilience, transparency, and user ownership.

The implications of our research extend to developers, businesses, and end-users who seek to harness the transformative potential of blockchain technology in web application development. Our insights contribute to the growing knowledge in the field of decentralized computing, opening avenues for further advancements in utilizing canisters for innovative web-based solutions.

Contents

Chapter 1

Introduction

By shedding light on the transformative potential of canisters and blockchain technology, this research aims to inspire innovation, facilitate secure and transparent digital transactions, and contribute to the advancement of blockchain and web application development as a whole.

This thesis seeks to address these challenges by exploring the integration of blockchain technology, specifically through the use of canisters, into web application development. Canisters, as self-contained units of computation on the Internet Computer, offer a unique framework for building decentralized applications. By leveraging the power of canisters, developers can create web applications that benefit from the security, transparency, and decentralized nature of blockchain technology.

1.1 Aim of the Project

With the wide popularity of Internet and related technologies, various Industry 4.0-based applications have been used across the globe in which sensors and actuators sense, compute and communicate the data for industry automation. As in Industry 4.0-based applications, data between different locations flows using an open channel, i.e., Internet, so threats to security and privacy has also increased manyfold. Such applications deal with data in large volumes and hence, so it is necessary to consider issues such as-data heterogeneity, data integrity, and data redundancy alongwith the security and privacy concerns. Moreover, different applications require datasets from different domains in different formats. Therefore, it is also needed to standardize the data format so that it can be used by different Industry 4.0-based applications.

1.2 Motivation

In today's digital landscape, the potential of blockchain technology to transform various industries has captured significant attention. Blockchain's decentralized and immutable

nature offers new possibilities for secure and transparent transactions, data management, and trust-building mechanisms. However, despite its promises, the adoption of blockchain in practical applications still faces challenges, particularly in the realm of web application development.

Traditional web applications often rely on centralized architectures, which can be vulnerable to single points of failure, data breaches, and lack of transparency. Additionally, scalability and performance limitations have hindered the widespread adoption of blockchain in web development. These challenges have sparked the need for innovative approaches that combine the benefits of blockchain technology with efficient web application design.

The motivation for this research lies in the potential of canisters to revolutionize web application development. By incorporating blockchain principles and utilizing canisters as autonomous computational units, developers can overcome the limitations of centralized architectures. Canisters provide a scalable, secure, and tamper-resistant environment that enables the implementation of innovative web applications with enhanced privacy, data integrity, and user control.

Furthermore, this thesis aims to contribute to the existing body of knowledge by investigating the practical implications, challenges, and opportunities associated with utilizing canisters in blockchain-powered web applications. By examining real-world use cases and evaluating the performance and scalability of canister-based web applications, this research seeks to provide valuable insights for developers, businesses, and researchers interested in harnessing the potential of blockchain technology.

Ultimately, the findings of this research endeavor can pave the way for the widespread adoption of blockchain-powered web applications, transforming industries such as finance, supply chain, healthcare, and beyond. By bridging the gap between blockchain technology and web application development, this thesis aims to empower developers and organizations to embrace the decentralized paradigm and unlock the full potential of blockchain in the context of web applications.

1.3 Thesis Organization

To address the aim of our project and delve into the motivations behind it, this thesis is organized as follows:

Chapter 2: Background

In this chapter, we lay the foundation by providing a comprehensive overview of the background information related to our research. We explore the architecture of blockchain technology, including important concepts like the Merkle hash tree and Elliptic Curve Cryptography. We also dive into the double spending problem, showcasing its real-world illustration, examples, and existing solutions. Additionally, we discuss the diverse

applications of blockchain, such as its impact on transportation, climate change, and healthcare. Furthermore, we explore smart contracts, explaining how they work and highlighting prominent platforms like Ethereum and Canister. Finally, we touch upon decentralized applications (DApps) and their significance in the blockchain ecosystem.

Chapter 3: Methodology

This chapter outlines the approach we followed to achieve the objectives of our project. We begin by describing our careful selection process of the platform we worked with. We then provide insights into the system design phase, where we conceptualized and planned the technical aspects of our solution. Additionally, we detail the implementation process, discussing the practical steps we took to bring our solution to life. This chapter offers a comprehensive view of the methodologies employed throughout our research.

Chapter 4: Experimental Results

Here, we present the results obtained from the experiments we conducted as part of our research. Our focus is on evaluating the transaction speed, a crucial metric that helps us gauge the efficiency of our proposed solution. By analyzing the experimental data, we gain insights into the performance and effectiveness of our system.

Chapter 5: Conclusion

In this final chapter, we summarize the key findings and contributions of our research. We provide a concise overview of the entire thesis, highlighting the implications of our results. Additionally, we identify areas for future work and improvements, specifically addressing the need to enhance the update call performance and simplify the deployment of web applications on Canisters and the Internet Computer. This chapter concludes our thesis, emphasizing the significance of our research and its potential for future advancements.

Chapter 2

Background

The main driver to the adoption of Blockchain technology was the ability to solve the double spending problem while maintaining the anonymity and privacy of the transacting user's information. Double spending is a situation in which a user of a digital currency can spend several times the same amount of money before there has been a realization that the amount has already been spent/claimed.

2.1 Blockchain Architecture

Industry 4.0 applications require careful consideration of security and privacy to prevent unauthorized access or data breaches. Without a strong security system, the system and data are vulnerable to various types of attacks that can compromise data confidentiality and integrity, and disrupt the overall functioning of the system. These attacks include Distributed Denial of Service (DDoS), Address Resolution Protocol (ARP) spoofing attacks, data rate alteration, network congestion, manipulation, noise interference, phishing, and config threats. It is crucial to prioritize prevention over reactive defense mechanisms and ensure compliance with legal rules to guarantee confidentiality, integrity, and privacy. The literature suggests that as Industry 4.0 automation increases, so does the risk of new types of cyber-attacks. Therefore, access control, authorization, confidentiality, availability, and integrity are essential concerns in Industry 4.0.

The blockchain technology has the potential to enhance security by eliminating the need for a centralized authority to perform operations. Instead, multiple users participate in verifying and validating transactions. The technology uses a distributed database structure that stores encrypted data from all nodes and verifies them using checks like Merkle Hash Tree (MHT) and Elliptic Curve Cryptography (ECC). Although there is a risk of database crashes or corruption, the linking of transactions using cryptographic keys and immutable ledgers makes it challenging for attackers to manipulate or delete information. Timestamps, public audit, and consensus mechanisms ensure data is stored in an immutable manner. The use of these mechanisms makes the security architecture robust and ensures data integrity and privacy.

2.1.1 Merkle Hash Tree

Ralph Merkle is the namesake of Merkle trees [1], which are utilized in cryptocurrencies like Bitcoin to efficiently and securely encode and encrypt blockchain data. These trees are also known as hash trees and are created through a process of repeatedly hashing pairs of nodes until only one hash, known as the root hash or Merkle root, remains. The construction of Merkle trees is done using a bottom-up approach. Each transaction is hashed, and then each pair of transactions is concatenated and hashed together, continuing until there is only one hash for the entire block. Figure 2.1 shows a simple binary Merkle tree.

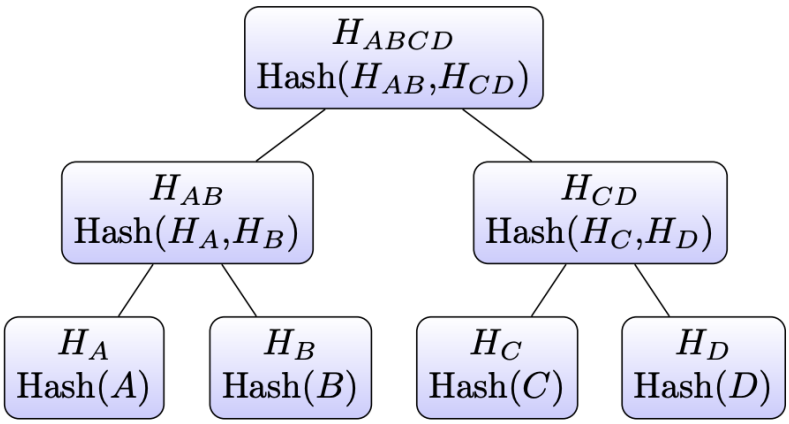


Figure 2.1: Simple binary hash tree.

The Merkle tree allows us to efficiently prove that some piece of data was used to generate a root hash without needing to have access or store all the original pieces of information. The proof size (Merkle proof) scales logarithmically with the number of data blocks as opposed to simply concatenating and hashing all the data at once, which would require storing large amounts of data. For example, in Figure 2.1, if we wanted to check if the piece of data A was used to generate the root hash, we would only need Hash of B (HB) and Hash of CD (HCD), because the latter summarizes data C and D. If any of the data changes, the root hash will also change. This may be extended to also authenticate large databases of potentially unbounded size by all computers large and small, without compromising on scalability and avoiding centralization of the services.

2.1.2 Elliptic Curve Cryptography

Victor Miller and Neal Koblitz independently introduced the concept of elliptic curve ciphers in the mid-1980s. These ciphers work similarly to other public-key cryptosystems, but they use operations on elliptic curves instead of modular arithmetic. The security of ECC, like all public-key cryptosystems, depends on difficult mathematical problems.

One such problem is the elliptic curve discrete logarithm problem, which involves finding an integer k given two points G and Y on an elliptic curve such that Y = kG (i.e., Y is G added to itself k times). Currently, methods for calculating discrete logarithms of elliptic curves are much less efficient than traditional methods of factoring or calculating discrete logarithms.

Example

We consider here that the secret keys KA and KB are generated randomly by sender (A) and receiver (B). Therefore, the randomly generated keys KA and KB are given by:

$KA = 1b66c808e6b5be6d6620934bc6ffa2b8b47f9786c002bfb06d53a0c27535641a5d1$

$KB = 1c7d15195432d1ac7f38aeb$
$054d07d9b2e1faa913b7d08a4d5efdd4a1ee8d9a31916d53a0c27535641a5d0d1$

Let us assume that (A) and (B) pre-agreed with the point Q given by:

Q = (0xd458e7d127ae671b0c330266d246769353a012073e97acf8,
0x3259305sfgr211f446bddc050cf7fb11b5673a1645086df3b)

When (A) sends the point $X = KAQ$ to (B) and (B) shares the point $Y = KBQ$ with (A), then the generated secret key is shared between (A) and (B). This secret key is common for both users and is given by:

$KS = 0x94f5a1cf2ed1dbb4322178df6bb4dd742c541884618b2989a3e5e66319667a640$

The elliptic curve which is being used for the Elliptic Curve Diffie-Hellman (ECDH) calculations is a 256-bit named curve brainpoolP256r1 (which uses a Diophantine equation for the generation of points). The private keys are randomly 256-bit (64 hexadecimal digits). The public keys and shared keys are 257 bits (65 hexadecimal digits, 256 bits due to key compression). Due to randomization, the secret keys KA and KB are different, but the calculated shared secret key between (A) and (B) will always be the same.

2.2 Double Spending Problem

The primary motivation for the acceptance of Blockchain technology was its capability to resolve the issue of double spending without compromising the confidentiality and privacy of user information during transactions. Double spending refers to the circumstance where a digital currency user can spend the same amount of money multiple times before it is detected that the amount has already been utilized or claimed.

2.2.1 Illustration

Problem starts with when user requests a transaction , miner picks the transaction and solve complicated problem to add block to blockchain as long as miner spends all his currency and sends this information to the real blockchain but not to his own isolated chain.Then, the corrupted miner solve problem faster than real miners (they are needed to verify the block and add it to blockchain) and he make an isolated blockchain bigger than the real one. As a result, the democratic governance rule states that the blocks will add to the larger one by removing the previous records that they have ,so in the new isolated chain, the corrupted miner would be able to spend all of the currencies that he had spent once in the real blockchain.

2.2.2 Examples

As for Table 2.1, Blockchain networks are not completely immune to attacks, with common examples including Wallet Attacks, Double Spending Attacks, DDoS Attacks, Border Gateway Protocol (BGP) Hijacking, Spam Attacks, Decentralized Autonomous Organization (DAO) Attacks, and Selfish Mining Attacks. These attacks can result in theft of funds or disruption of network operations. To mitigate these threats, blockchain networks use security measures such as encryption, multi-factor authentication, and consensus algorithms, but continuous monitoring and updates are necessary to ensure network and user protection.

Table 2.1: Examples of attacks to blockchain

No	Attack Name	Targeted Area	Effect of Attack	Possible Countermeasure found
1.	Brute Force Attack	Computing Power, Pow Consensus	Data encryption	inserting observers in the network, notify the merchant about an ongoing double spend
2.	Refund Attack	Payment protocol	Lose money, reputation	publicly verifiable evidence
3.	Wallet Attack	Private key	Lose of bitcoin	threshold signature based two-factor security, hardware wallets , Password-Protected Secret Sharing (PPSS)
4.	Time Hijacking Attack	Network	Fake peers	constraint tolerance ranges, network time protocol (NTP) or time sampling on the values received from trusted peers
5.	Long Range Attack	Database	Alter transaction history	Nodes trust identity provider, implementation of trusted hardware
6.	BGP Hijacking	Database, Protocol	Fake transaction	Human driven process consisting of altering configuration or disconnecting the attacker.
7.	Sybil Attack	Network	Pseudonymous identities, threatens user privacy	Xim (a two-party mixing protocol)
8.	DDOS Attack	Network	Generates huge unnecessary responses about transaction	Proof-of-Activity (PoA) protocol
9.	Eclipse Attack	Network	inconsistent view of the network and blockchain	Use whitelists, disabling incoming connections
10.	DAO Attack	Computing Power	Fake transaction	Hard fork proposal, Soft fork proposal
11.	Nothing at Stake Attack	Block	Slow consensus time	Slasher Protocol
12.	Pool Mining Attack	Block, Computing Power	Slow verification time, fake transaction	Not Found
13.	Double Spending Attack	Bitcoin transaction, Pow Consensus	lose products, create forks	Recipient oriented transaction
14.	Selfish Mining Attack	Block, Computing power	Increase personal share on transaction	Address bitcoin protocol and raise threshold, computing branches are same length and propagate all of them, Zero Block technique

One common attack is the Wallet Attack (Wallet), which involves hacking into a user's wallet and stealing their private keys. Once the attacker has access to the private keys, they can easily transfer the funds to their own wallet. Double Spending Attack (Double Spending) Attack is another popular attack where the attacker attempts to spend the same coins twice, causing a loss of funds to the network.

DDoS Attack is also a potential threat to blockchain networks. This attack aims to overload the network with a large volume of traffic, which can cause the network

to become slow or even unresponsive. BGP Hijacking is another type of attack where attackers redirect the network's traffic to a malicious server in order to carry out their nefarious activities.

Spam Attack (Spam) Attack is another attack where attackers send a large number of transactions to the network in order to overload the network's capacity. The DAO Attack is a sophisticated type of attack that occurred in 2016, which involved exploiting a vulnerability in a smart contract. This attack resulted in the loss of millions of dollars worth of cryptocurrency.

Finally, Selfish Mining Attack (Selfish Mining) Attack is a type of attack where a group of miners try to monopolize the mining process by withholding the blocks they have mined from the network and publishing them later. This allows them to gain an unfair advantage over other miners and can lead to centralization of the network.

2.2.3 Double Spending Probelm Solutions

Smart contract applications are similar to web applications that run over the blockchain. Like web application bugs, they also comprise errors, however, these bugs can lead to serious challenges. For example, the DAO was able to raise 150m whilst the attacker was able to steal about 60m due to code bugs. Rubixi and GovernMental are some of the smart contract applications which had flaws due to code bugs. Application bugs may not only allow attackers to steal money, but also influence an application to function differently.

- Mythril

  Mythril is a tool used to enhance the security of smart contracts written in Solidity by detecting any potential errors in the code. As an open-source tool, Mythril employs symbolic execution technique to identify potential security vulnerabilities. To analyze security flaws, Mythril executes smart contract bytecode in a specialized Ethereum Virtual Machine (EVM) and employs four significant stages to achieve its security analysis. In the event that a program flaw is discovered, Mythril conducts an analysis of input transactions to establish the possible reasons behind the flaw. This security approach assists in identifying the underlying cause of the program's vulnerability, and as a result, reducing the likelihood of its exploitation. Additionally, if a developer provides the source code of the contract, Mythril is capable of locating and identifying any bugs in the code.

  Mythril is an excellent tool for analyzing smart contracts built for EVM bytecode. It can detect security vulnerabilities in Ethereum, Hedera, Vechain, Tron, Quorom, Roostock and basically every or any EVM compatible blockchains. Mythril is exclusively based on symbolic execution engine. It thoroughly goes through all probable states within a contract with the help of function(s) call. Mythril analysis uses the bytecode of the smart contract to decompile it back into EVM opcode instructions. It then explores all probable program states over 'n' transactions. Mythril

uses LASER, Symbolic Virtual Machine (SVM) to create an environment, where it executes opcode and reveal vulnerabilities. LASER computes all probable program states. Mythril uses a number of analysis modules to determine if the vulnerability subsists. If a compromised or vulnerable state subsists. Mythril uses Z3, an automated theorem prover, to prove or disprove the extendibility of the state. Z3 is delivered by Microsoft Research under Massachusetts Institute of Technology (MIT) license. All the issues about symbolic execution engines, detecting vulnerabilities are well documented.

– Mythril installation / Usage

a. Environment preparation by installing Homebrew, python and python3.

b. Install Mythril using following commands from Figure 2.2.

```
>brew update
>brew upgrade
>brew tap
ethereum/Ethereum

>brew install leveldb
>brew install solidity
>pip install mythril
```

Figure 2.2: Mythril Installation commands

c. Checking Mythril for proper running ¿myth -x smartcontract.sol‘.

- ZEUS

ZEUS ensures both reliability and scalability by using three main observations. Firstly, even though the blockchain operates like a concurrent system with task-based semantics, a transaction consists of only one call chain that starts from a publicly visible function in the smart contract. This means that verifying most properties requires exploring a much smaller state space. Secondly, smart contracts are driven by both control and data. Therefore, ZEUS models contracts using abstract interpretation, which computes loop and function summaries over data domains. These summaries are used during the model checking phase, which operates on a reduced state space. Lastly, Constrained Horn Clauses (CHC)s are used to represent verification conditions, which can be easily solved by Satisfiability Modulo Theory (SMT) solvers.

– Implementation
In order to develop a tool for translating Solidity smart contracts to Low Level Virtual Machine (LLVM) bitcode and detecting bugs, we utilized the of the smart contract derived from the Solidity compiler solc, and implemented both the policy builder and the translator in C++. The policy builder required approximately 500 lines of code, while the ranslator, including the LLVM passes for bug detection, required approximately 3000 lines of code.

For ease of implementation, we leverage Seahorn as our symbolic model checking backend for verification of policies. Instead of building the verifier from scratch, we determined that Seahorn provides us with an off-the-shelf implementation of generating verification conditions using CHCs over LLVM bitcode. Furthermore, use of existing tools that have been tested for bugs and fine tuned for performance, both of which are critical for verifiers, helps reduce ZEUS's false alarms and improve verification times. However, as will be shown later in § VI-D, ZEUS is not tied to Seahorn; it can be used with any other verifier that operates upon LLVM bitcode, such as SMACK or DIVINE .

- Blockchain-Based Sybil Attack Detection

As shown in Figure 2.3 ,In order for a sensor node to join the network, it must request access from the Cluster Head (CH) and provide its unique ID. Once granted access, the sensor node will periodically update its cluster membership information to account for any changes in its position or range. The CH will use a trust management model to evaluate the trustworthiness of each sensor node and generate a transaction containing the node's ID, location, and trust value for addition to the blockchain. If a sensor node relocates to a new cluster, it must request access again, and the corresponding CH will check the blockchain for previous information on the node. Each CH includes a timestamp for each period to detect any Sybil nodes. If a node is found to be a Sybil node, it is considered malicious and denied access. Otherwise, it is considered legitimate.

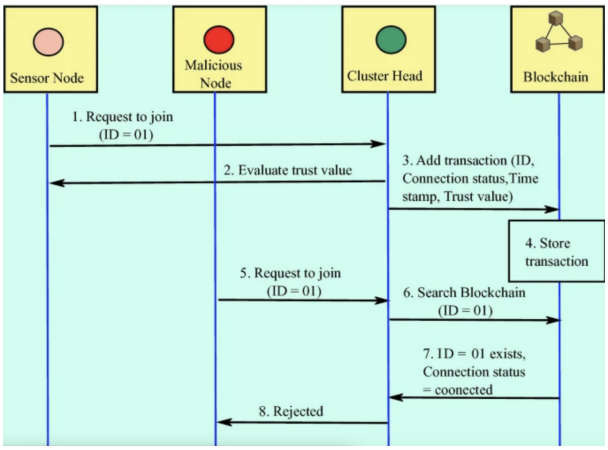


Figure 2.3: Blockchain-Based sybil detection model

– Trust model To estimate the trustworthiness of sensor nodes in the dynamic underwater channel, aHidden Markov Model (HMM) is employed. The HMM is capable of handling the changing nature of the network. The HMM considers packet loss and errors as observable states, while trustworthiness and maliciousness are hidden states. By analyzing the observable states, the HMM can effectively determine the level of trust or distrust associated with each sensor node.

- Pikachu Protocol(LRA)

  As Bitcoin does not allow for stateful smart contracts, we use an aggregated public key to represent the configuration of validators Ci in the Proof-of Stake (PoS) system. When the set changes significantly enough to configuration Ci+1, the aggregated key must be updated in the Bitcoin blockchain. This is done by having a transaction transferring the funds associated with the aggregated key of the previous validators Ci to the new aggregated key controlled by validators in configuration Ci+1. Instead of having each validator in Ci send a transaction to the Bitcoin network, this transaction is signed interactively, off-chain, and all the signatures are aggregated into one constant-size signature. Furthermore, we store the Merkle root of the state of the checkpointed PoS blockchain in the Bitcoin OP-RETURN field of the transaction from Ci to Ci+1. We store the data pertaining to this checkpoint off Bitcoin blockchain. While the data pertaining to the checkpoint could be stored anywhere (e.g., IPFS [2]) and validated against the state root stored in the Bitcoin transaction — our implementation uses a content-addressable key-value store implemented on top of the PoS system to store the actual checkpointed state. Figure 2.4 illustrates the high-level protocol. We note that since our work is based on Schnorr threshold signatures and uses Bitcoin's Taproot, it could be of independent interest to any project looking to implement threshold signing transactions on Bitcoin (for example, sidechains [3]).

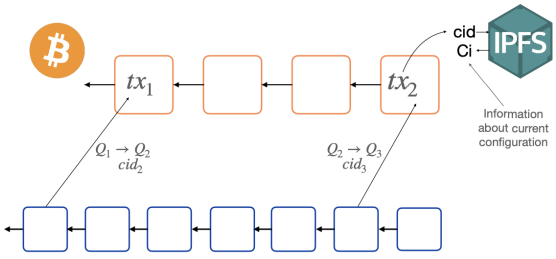


Figure 2.4: Pikatchu protocol model

2.3 Applications of Blockchain

Blockchain technology has come a long way since its inception as the underlying technology for Bitcoin. Today, it has found applications across various industries, ranging

from finance to healthcare, logistics to energy, and more. Blockchain's ability to provide decentralized, transparent, and immutable records make it a promising solution for industries looking to improve their operations and increase efficiency.

In finance, blockchain is being used for digital identity verification, cross-border payments, and to enable faster and more secure settlements. Healthcare is another industry that can benefit from blockchain's tamper-proof record-keeping and secure sharing of medical data among healthcare providers. In logistics, blockchain can help in supply chain management, by providing a transparent and secure record of goods as they move across various stages of the supply chain.

Energy is yet another industry where blockchain is being explored for its potential to enable a more efficient and decentralized energy system. Blockchain can facilitate peer-to-peer energy trading, where energy producers can sell their excess energy to consumers directly, bypassing the need for intermediaries.

These are just a few examples of the many applications of blockchain technology as shown in Figure 2.5. With its potential to increase transparency, reduce fraud, and lower costs, blockchain is likely to continue to disrupt industries and create new opportunities for innovation.

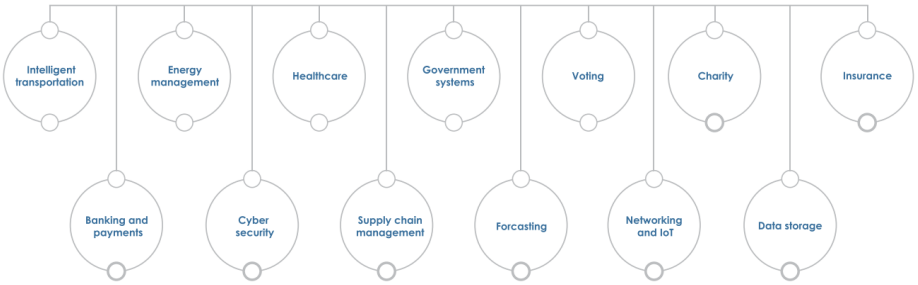


Figure 2.5: Blockchain applications

One of the most promising applications of blockchain technology is in the realm of financial services. Blockchain-based platforms have the potential to revolutionize traditional banking by providing faster, cheaper, and more secure transactions. It can also enable new types of financial services, such as peer-to-peer lending, microfinance, and remittances. Another area where blockchain technology can make a significant impact is supply chain management. By using a blockchain-based platform, it is possible to track goods from their origin to their destination, ensuring transparency and reducing the risk of fraud. This can improve efficiency, reduce costs, and increase trust between parties. Blockchain technology can also be used in voting systems to enhance transparency, security, and verifiability. By using a blockchain-based platform, it is possible to prevent tampering, ensure anonymity, and allow for the secure storage and retrieval of votes. Other potential applications of blockchain technology include identity management, healthcare, real estate, gaming, and energy management. Overall, the potential

applications of blockchain technology are vast, and it is likely that we will see more innovative use cases emerge in the future.

2.3.1 Blockchain in Transportation

The Blockchain Transportation and Logistics Framework (BCTLF) is an intelligent framework for transportation and logistics that uses blockchain and Internet of Things (IoT) technology. Unlike traditional vehicular communication systems, which are susceptible to privacy breaches that compromise the security of messages containing sensitive information such as the identity of the driver, type of vehicle, and location, the BCTLF offers a solution. To address this issue, a common approach has been to use pseudonyms to conceal the data. However, studies have shown that the expense of maintaining these pseudonyms can escalate, and distributing certificates can be challenging.

Smart vehicles and electric cars are equipped with advanced technology including embedded computers, GPS systems, short-range wireless network interfaces, and access to in-car sensors and the internet. To ensure that sensitive data such as driver and vehicle identity, cryptographic keys, location, predicted travel direction, traffic and road congestion is shared and stored safely, a secure implementation of sharing and storing must be established. This requires a fully or semi-distributed vehicular network topology to avoid security vulnerabilities and bottleneck issues that may arise in centralized systems. The use of blockchain technology can be beneficial in managing data in the Internet of Vehicles (IoV) processing layer and security layer, which can lead to innovative traffic services and secure data transfer while protecting vehicular systems from security attacks.

As an application example, a simple selection mechanism of the charging unit for Electric Vehicle (EV) drivers is developed based on smart contracts. a scheduling approach for EV charging was proposed considering the battery capacity, the rate or behavior of charging and discharging operations, and the relative cost of charging. This mechanism could be extended to consider other selection criteria, such as actual battery status, traffic congestion, and service delay. Other research works focused on designing efficient energy trading frameworks for EVs. In [4], an optimized cost-aware trading energy platform was studied over a consortium Blockchain. This platform applied a contract-based incentive mechanism to respect the preferences of EVs and improve their participation cycle.

The payment research field concentrates on investigating new and advanced payment and billing solutions for users of the IoV. These solutions are aimed at ensuring secure and efficient transactions while also optimizing energy consumption and pricing. In the IoV processing layer, there is a suggestion for a Blockchain-based billing service that guarantees secure transactions between electric vehicles EVs and charging stations. Similarly, a different study recommends a unique transaction structure for Hyperledger systems that allows for trustworthy and tamper-proof payments. Moreover, researchers have developed an optimization model for the distributed scheduling mechanism of EV battery swap stations, which helps to decrease power generation cost and load. In the

IoV communication layer, a new configuration of EVs and charging stations has been created to personalize Bitcoin transactions in a private network, which leads to reduced costs and verification delay.

2.3.2 Blockchain Role in Climate Change

In recent times, global warming has become an increasingly urgent issue, and its impacts are being felt in many countries. The primary cause of this phenomenon is human-generated greenhouse gas emissions, which could result in an increase of 2.8○ by 2050 if no corrective measures are taken in the current decade. blockchain technology could play a role in addressing this issue by limiting human activity. Deforestation, caused by factors such as factory and livestock farming, results in reduced CO2 absorption and the buildup of CO2 in the atmosphere. Blockchain technology can reduce paperwork and streamline digital applications, thus minimizing human activities that contribute to this problem. By implementing blockchain by 2030, global warming could potentially be reduced by at least 4○ by 2050. Additionally, the authors identified several limitations of the Reducing Emissions from Deforestation and Forest Degradation (REDD+) project and proposed solutions based on blockchain technology, such as cryptocarbon management, green finance, and sustainable investments.

There are many climate-conscious blockchain initiatives in various stages of development. SolarCoin, for example, uses a blockchain platform to incentivize solar-energy producers by rewarding every megawatt-hour of electricity they produce with one free SolarCoin. This digital reward can be used as a medium of exchange or converted to any other currency. Projects such as Earth Dollar aim to link carbon credits (pollution permits that are issued for emissions avoided elsewhere) to blockchain tokens (representations of a particular asset or utility within the platform). Some initiatives are enabling automated smart-contract payment protocols, so that embodied carbon emissions from consumer purchases can be calculated and carbon credits purchased automatically. Infinite Earth's Veridium Labs, for example, a HongKong-based private company working in partnership with IBM, are connecting their payment system (VerdePay) with carbon credits produced from Infinite Earth's Forest reserve in Rimba Raya, Central Kalimantan.

2.3.3 Blockchain and Health Care

Blockchain technology can be utilized effectively in the healthcare industry to store sensitive patient data such as medical records, images, videos, and documents. These data require high levels of security and privacy while also being readily available to authorized users. By implementing blockchain, the security and privacy of data can be maintained. A healthcare model based on Bitcoin can store health data from individual users, but the storage management system is not bandwidth-intensive, and network resources are dispersed through throughput optimization. A better solution is to use an access control

manager and store instances in a database management system. Authorized administrators can access the encrypted and time-stamped records using a unique identifier that locates them in the storage device. All the data stored in the blockchain database are referred to as data lakes, which are crucial for data.

Blockchain technology has gained popularity in healthcare, especially in the Management of Electronic Medical Records (EMRs). EMRs, also known as Electronic Health Records (EHRs) or Personal Health Records (PHRs), refer to patients' medical or health-related data that are electronically created, stored, and managed. A significant amount of research has been devoted to the use case of blockchain in managing EMRs, with almost half of the selected papers (32 out of 65) addressing this topic. The decentralization, immutability, data provenance, reliability, robustness, smart contracts, security, and privacy features of blockchain are being highlighted as reasons why it is suitable for storing and managing EMRs. Some studies are focused on facilitating patient-centric data sharing among various healthcare stakeholders, such as providers, researchers, and insurers. In line with the European General Data Protection Regulation (GDPR), which requires explicit consent from patients for the processing of sensitive personal data, blockchain is widely proposed as a technology that can empower patients to control how their data are shared, processed, or used within the healthcare platform.

Figure 2.6 illustrates the several on-ground industrial representatives of Blockchain capabilities to successfully implement healthcare culture perspectives and overall development. There have been various associated industrial/medical-care supporters or providers, which helps carry out the research and investigations for realising the Blockchain practices in healthcare and its core domains, too [5][6]. These observed pro viders BurstIQ, Guardtime, Robomed, Simply vital, Encrypgen, Chronicled, Tieion, etc., are the few agencies supplying and favouring the practising of Blockchain technology at ground levels.

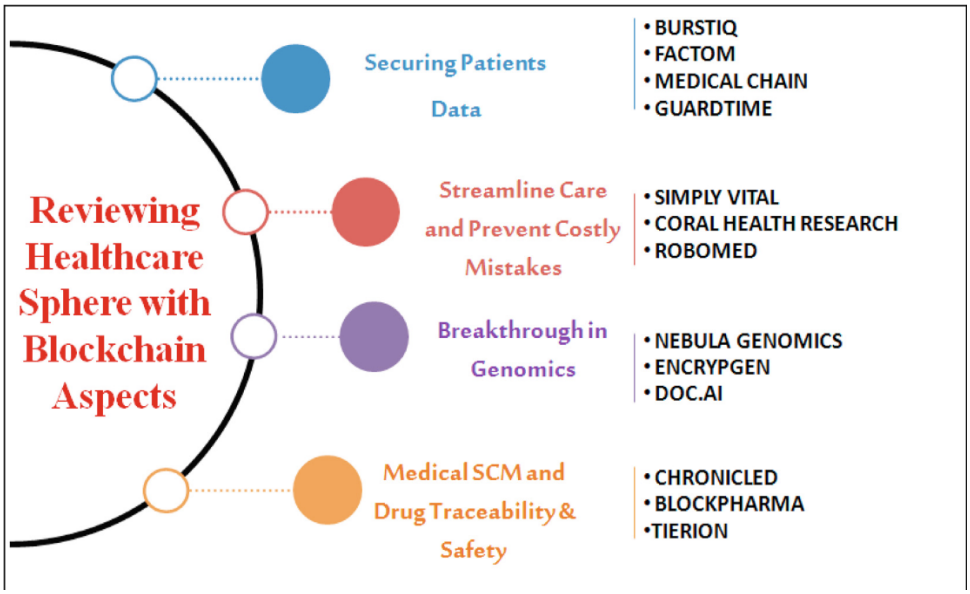


Figure 2.6: Enablers of blockchain implementation in healthcare services.

Guardtime, a company that utilizes a blockchain-powered platform to ensure the security of more than 1 million patient records in Estonia, is recognized in various reviews as a prominent example of blockchain implementation in EMRs management [7][8]. Another notable illustration is the MedRec project [9], jointly developed by MIT Media Lab and Beth Israel Deaconess Medical Center. This project aims to empower patients by granting them control over their own data, allowing them to determine who can access it through fine-grained access permissions built on the blockchain. Additionally, the Gem Health Network (GHN), developed by the US startup Gem on the Ethereum blockchain platform, enables different healthcare practitioners to have shared access to the same data [9]. Healthbank, a Swiss digital health company, is also actively engaged in empowering patients to have complete control over their data using blockchain technology [9]. In another publication [10], the author discusses the Medicalchain project, which utilizes a blockchain-based platform to facilitate the sharing of patients' medical records across international healthcare institutions. The author also mentions the Healthcoin initiative, which aims to construct a global EMRs system. Various other entities such as Factom, HealthCombix, Patientory, SimplyVital, IBM's Watson, BurstIQ, Bowhead, QBRICS, and Nuco are actively involved in different initiatives and projects that leverage blockchain for patient-centric EMRs systems [10].

2.4 Smart Contracts

Smart contract technology is a digital transaction protocol that operates on its own to enforce the conditions of a contract and establish an understanding between multiple parties. Smart contracts are characterized by their disintermediation and transparency, which facilitate commercial efficiency, decrease legal and transaction costs, and enable anonymous transactions. Their many advantages make them a highly sought-after technology, particularly in the financial industry where they minimize the risk of non-payment and fraud while enhancing the quality of financial contracts with a level of trust that is independent of intermediaries.

Although technology supporting smart contracts has seen numerous advancements,their application in various companies remains poorly understood. Businesses are facing new challenges concerning system security and information technology, as well as trust in data transaction systems and transparency in data exchange between different organizations. The decentralization of business processes and workflow within organizations poses a further challenge in meeting the technological requirements of companies that aim to standardize data transaction security, trust, and transparency. To address these organizational data transaction issues, smart contract technology based on blockchain is suggested as a potential solution for companies.

2.4.1 Principal of Operation

According to [11], we introduce a high-level overview of a smart contract's process, starting from its development until the verification of transactions. Figure 2.7 provides a

graphic representation of those different steps. Before diving into the smart contract's process, we give a definition of the different terms that will be used:

- Developer: an individual who implements the logic of a smart contract using a specific set of instructions compatible with or provided by a blockchain platform
- User: any entity that uses the services of a smart contract
- Transaction: a query made to a smart contract program
- Blockchain platform: the set of applications and protocols used to maintain and manage a blockchain
- Node: an entity having an account on the blockchain platform that can execute and validate transactions
- Faulty node: a node susceptible to submitting false results after the execution of a smart contract

The process of deploying a smart contract on a blockchain platform typically involves the following steps:

1. Developers write the smart contract logic using a supported programming language for the target blockchain platform. They then compile the source code using a specific compiler provided by the platform to obtain a byte code representation of the contract.
2. The compiled byte code is published to the blockchain platform, where it is stored on the blockchain. Depending on the platform, the smart contract program may be read-only or modifiable. For example, Ethereum does not allow modifications to smart contracts [12], while EOSIO allows overwriting by uploading new byte code [13]. If the contract is read-only, developers need to publish a new version to provide updates. The initial state of the smart contract represents the initial values of its internal variables.
3. Access to a published smart contract depends on the blockchain platform. In Ethereum [12] and Neo [14], the platform provides an address that developers can use to interact with the contract. In EOSIO [15], the contract is published to a previously created account, and the account's identifier is used for access. Once users obtain the address or identifier, they can send transactions to the contract, specifying the desired function and its arguments. If required, the transaction may include an amount of platform currency to initiate the function's execution. The transaction is stored in the platform's transaction pool for execution and validation. Step 3 in Figure 2.7 is based on the functioning of [12][14].

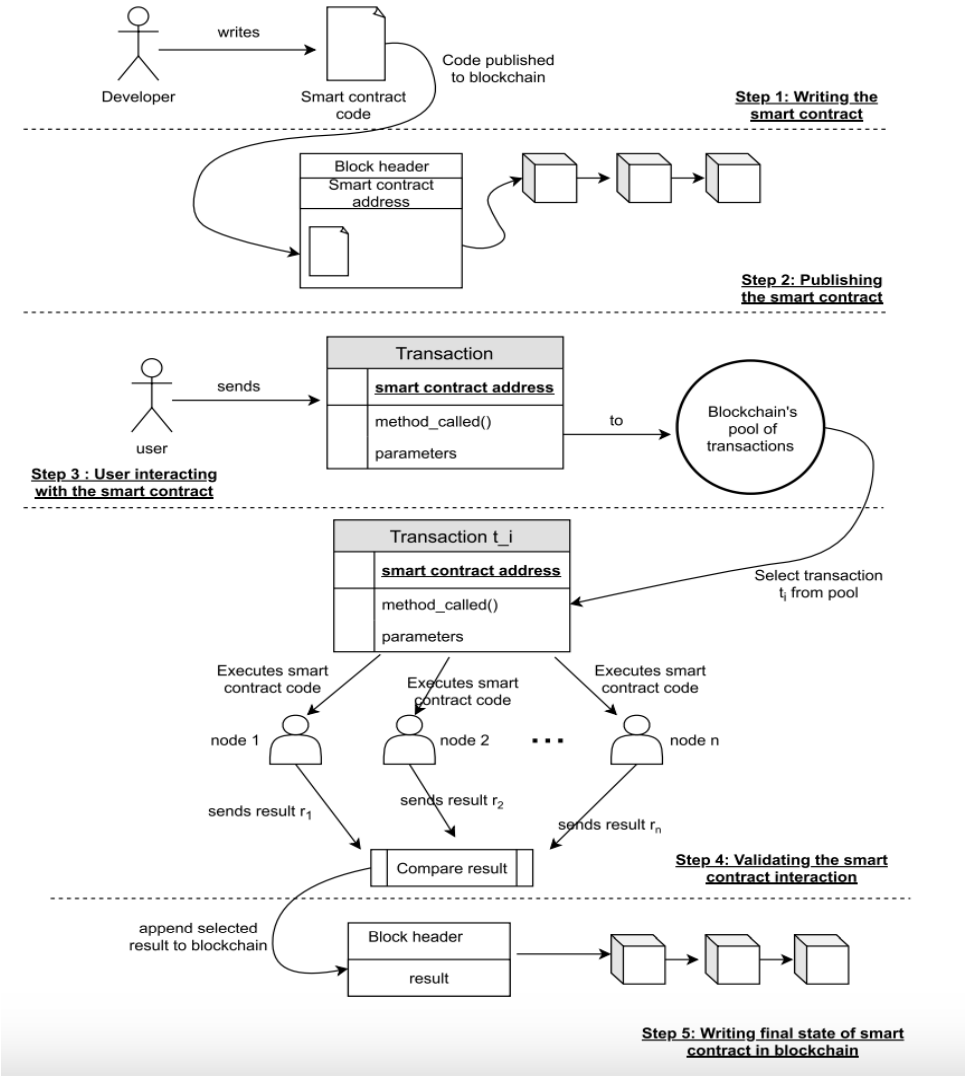


Figure 2.7: Overview of smart contracts.

4. From the transaction pool, the blockchain platform selects a set of transactions to be executed and validated. During the execution phase, the specified functions in the transactions are executed by a set of nodes. In the validation phase, the nodes compare their results and reach a consensus on the valid result according to the chosen consensus protocol. For example, in a Byzantine Fault Tolerant (BFT) [16] consensus protocol, a certain number of nodes are selected to execute and validate transactions, and the result agreed upon by the majority of nodes is considered valid. Other consensus protocols include proof-based consensus, where each node must provide a valid proof of execution [17], and hybrid protocols that combine BFT and proof-based mechanisms [18][19].

5. Once the valid result is determined, it is inserted into a block that is appended to the blockchain. Additionally, the initial state of each smart contract specified in the

executed transactions is updated. If a validated transaction modified the internal variables of a smart contract, the new values become the initial values for future transactions involving the contract.

Source Code and Bytecode

The code for a smart contract is commonly written in Solidity, a language that shares similarities with JavaScript in terms of syntax. Solidity smart contracts have a structure resembling that of a class in object-oriented programming. Figure 2.8 provides an illustrative example. Comparable to the Java compiler, the Solidity compiler generates bytecode from the source code, which is then executed by the EVM. The Ethereum bytecode consists of various opcodes, which are low-level instructions. It is essential to note that only the bytecode of a smart contract is stored in Ethereum. Additional information regarding Solidity and its bytecode format can be found in the official Solidity documentation [20].

Figure 2.8: An example of a smart contract written in Solidity.

Subcontracts and Libraries

In order to enable separation of concerns, the Solidity language provides subcontracts and libraries constructs. Similarly to nested classes (as in object-oriented programming),

subcontracts and libraries have each a specific purpose. Subcontracts enable developers to establish object-oriented relationships between contracts, such as inheritance and interface implementation. Libraries often provide a set of utility methods that are mindful of corner-cases or optimize processing time (e.g., a library to perform mathematical operations without incurring into underflows and underflows). In the example depicted in Figure 2.8, there are two subcontracts (ERC20 and MyCoin) and two libraries (SafeMath and VectorSum).

2.4.2 Platforms

Bitcoin is recognized as one of the prominent blockchain technology products, driving ongoing advancements in research and technology commercialization. It serves as the groundbreaking catalyst for the emergence of blockchain 1.0 and the development of cryptocurrency. This technology exhibits key characteristics such as decentralization, resilience to damage, anonymity, and auditing capabilities [21]. Bitcoin operates on the foundation of the blockchain system. However, when it comes to implementing intricate contract logic, Bitcoin encounters a limitation. Due to the constraints of the Bitcoin scripting language, which encompasses a total of 256 instructions (with 15 currently disabled and 75 reserved), writing complex contract logic becomes impractical. Consequently, Bitcoin can be regarded merely as a rudimentary prototype of a smart contract. On the other hand, continuous efforts are being made to enhance smart contract functionality. Among the platforms focusing on this area is Ethereum. Ethereum was developed to address the deficiencies of the Bitcoin platform. It revolves around the central concept of executing user-defined programs on the blockchain, enabling the creation of sophisticated custom smart contracts using a Turing-complete programming language [22].

In the development of Ethereum, the Ethereum smart contract code is written in bytecode language based on the stack that runs on the EVM [23][24]. The high-level languages used in Ethereum development are Solidity1 and Serpent2 [23]. After that, the program code is compiled into EVM bytecode to run. With complex smart contract logic blockchain technology implemented, currently, Ethereum is the most popular platform for developing smart contracts; then, Ethereum is marked as the developer of blockchain technology which is known as Blockchain 2.0 [25]. As part of Ethereum, Hyperledger Fabric, Corda, and BigchainDB are several platforms that can be used to develop smart contracts [26]. One of the advantages of smart contracts is the implementation of protocols that work by consensus [27][28]. Smart contracts will work by encoding predetermined rules and carry out related operations when the agreed provisions have been fulfilled [29]. Smart contract implementations can be used in a variety of domains including smart assets [30] (e.g., Slock.it3, a German company that uses Ethereum-based smart contracts for leasing, selling, or sharing other transaction activities without the involvement of intermediaries) [30]. In addition, the implementation of smart contracts has been used in the implementation of self-government or autonomous arrangements (for example, digital property management such as ujomusic4, e-voting, and supply chain) [31].

1. Electronic Government Decentralized Autonomous Organization (eGov-DAO)

One application of blockchain and smart contracts in the government sector is eGov-DAO. eGov-DAO serves as a framework for smart contracts that enables the provision of e-government services through an automated and efficient system. Its integration ensures transparent service delivery [32]. The primary purpose of eGov-DAO is to facilitate real-time monitoring and analysis of government services while upholding principles such as transparency, accountability, and provision. A significant advantage of this system is its ability to improve the management of national resources [32]. By maintaining comprehensive audit records, eGov-DAO ensures transparency in court proceedings involving relevant parties. The user-friendly design of eGov-DAO minimizes the need for extensive user training [32].

The implementation of eGov-DAO is particularly beneficial in government contracts, which involve agreements between the government and vendors that are publicly recorded [32]. The previous system exhibited several weaknesses, including inefficient allocation of contracts and deals that required extensive interaction between institutions, leading to excessive human labor [33]. Furthermore, while the government invested significant human and financial resources in providing excellent services, it resulted in a lack of transparency in government administration and inefficient performance. Smart contracts offer a solution to enhance transparency, trust, cost reduction, and process simplification.

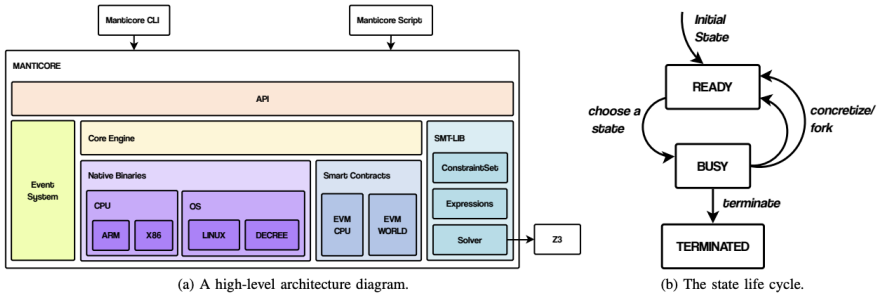
The eGov-DAO framework is a versatile blockchain framework applicable to government contract policies. It was specifically developed with the United States Small Business Administration policy in mind [32]. The framework's initial phase involves explaining smart contract and blockchain applications in the context of US contracts. It then introduces contract allocation rules mandated by the Small Business Administration policy, and the DAO framework incorporates these requirements and governs the allocation process within the smart contract. eGov-DAO defines system-related parties, system user activities, and establishes a model for the primary contract process. The final phase of eGov-DAO involves validating the agreed-upon rules within the smart contract [32].

2. Manticore

Manticore is a framework based on blockchain technology that employs a smart contract approach for symbolic execution in binary analysis. It is utilized by Trail of Bits and Defense Advanced Research Projects Agency (DARPA) Cyber Grand Challenge (CGC) as a symbiotic code testing solution for code assessment and program analysis. One of Manticore's strengths lies in dynamic symbolic execution, which involves exploring the semantic distance of program code. This analysis technique traverses each path, considering all code, and employs dynamic symbolic execution principles. This process entails identifying path predicates and setting constraints for program testing. Despite its advantages, Manticore has not been widely adopted in the industry due to the lack of a flexible and user-friendly tool. Furthermore, the existing framework is closely tied to

the traditional execution model, limiting symbolic execution to an alternative technique, such as seen in the Ethereum platform [34].

Manticore's design is highly flexible and supports both traditional computing environments (x86/64, ARM) and exotic ones, such as the Ethereum platform. To our knowledge, it is the only symbolic execution framework that caters to such different environments. It is also simple, extensible, and as self-contained as possible, avoiding unwarranted external dependencies. Figure 2.9 shows Manticore's architecture. The primary components are the Core Engine and Native and Ethereum Execution Modules. Secondary components include the Satisfiability Modulo Theories (SMT-LIB) module, Event System, and API.



(a) A high-level architecture diagram. (b) The state life cycle.

Figure 2.9: Manticore overview.

The primary function of Manticore is to serve as a technology for program testing using a symbolic execution analysis approach. It can be employed for symbolic execution on various execution platforms. In terms of testing program code, Manticore demonstrates comparable performance to other standard symbolic execution tools, providing an average code coverage of 66% specifically for smart contracts [34].

Over the past decade, symbolic execution has witnessed significant development and research interest, resulting in the creation of notable symbolic execution tools like KLEE, which is the foremost and widely adopted example of symbolic execution today. Additionally, several symbolic execution frameworks are currently under development, including Angr, Triton, binsec, and miasm. These platforms are renowned in the field of binary analysis and offer extensive symbolic execution functionality that finds wide application in commercial software auditing [34].

3. Smart Contract Online Detection Freamwork against Attacks (SODA)

Current methods for safeguarding smart contracts can be broadly classified into two categories: offline and online. Offline approaches involve analyzing the code of smart contracts for potential vulnerabilities, checking for accuracy, reverse engineering bytecode, and detecting malicious code. However, offline methods have limitations as they lack runtime information, which can result in the failure to detect and remove all vulnerabilities. On

the other hand, online approaches are intended to detect attacks targeting smart contracts or protect them from attacks after deployment. These approaches fall into two categories: those that add protection code into the source/bytecode of smart contracts and those that add protection code into the runtime of smart contracts (such as EVM). However, these methods have limitations such as size restrictions and gas mechanisms for bytecode protection, and they can be resource-intensive, leading to potential detection failures.

Smart contracts have attracted the attention of attackers due to their potential for significant cost savings. However, existing offline methods for identifying vulnerabilities or verifying smart contracts lack the capability to detect online attacks in real-time. Moreover, current online approaches are limited in their focus on specific attack types and cannot be easily extended to detect new forms of attacks. Developing a new online detection system for smart contracts from scratch is a time-consuming task that requires a deep understanding of blockchain internals, making the implementation of new attack detection mechanisms challenging [35].

As shown in Figure 2.10, SODA is integrated into a full node of any EVM-compatible blockchain, because only a full node can mine blocks. It separates the information collection and attack detection with layered design for the ease of developing detection apps. More precisely, SODA uses three modules, namely block collector, trans collector and ins collector, to collect the primitive information of blocks, transactions, and executed EVM instructions, respectively, from the smart contract layer, the consensus layer and the data layer of a full node. Note that EVM serves the data, consensus, and application layers [36], where the data layer contains the defined data structures and the data types (e.g., account, transactions), the consensus layer regulates how nodes reach consensus, and the smart contract layer stores smart contracts [36].

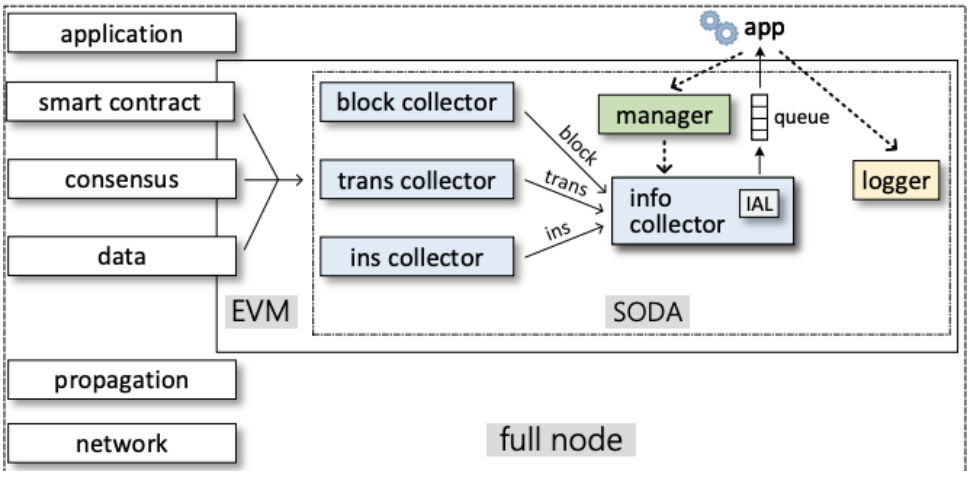


Figure 2.10: SODA architecture.

Introducing SODA, a novel and comprehensive online detection framework compatible with all blockchains supporting the EVM. SODA sets itself apart from existing online

approaches by offering enhanced capabilities, efficiency, and compatibility. Firstly, SODA provides users with a user-friendly platform for developing various attack detection applications, separating information gathering and attack detection through a layered design. At a higher layer, SODA presents a unified interface for creating diverse attack detection applications. At a lower level, SODA instructs the EVM to collect essential primitive information necessary for detecting a range of attacks and constructs 11 types of structural information to simplify application development. Leveraging SODA, users can swiftly develop new applications with minimal code modifications to the EVM. Secondly, SODA focuses on efficiency by implementing on-demand information retrieval to minimize additional costs associated with information gathering. Additionally, it employs dynamic linking to eliminate the overhead from inter-process communication. This design enables users to develop detection applications using any programming language capable of generating dynamic link libraries. Thirdly, as more blockchains adopt the EVM as the smart contract runtime, EVM offers seamless migration without requiring application modifications. To date, eight detection applications have been developed using SODA to identify attacks exploiting major vulnerabilities in smart contracts. SODA has been integrated into three popular blockchains: Ethereum, Expanse, and Wanchain. Extensive experimental results demonstrate the effectiveness and efficiency of SODA in various detection applications [35].

2.4.3 Ethereum

Ethereum is a software platform built on blockchain technology that enables the creation and execution of smart contracts and Decentralized Applications (DApps). It serves as the foundation for a virtual currency called Ether. Smart contracts on Ethereum are defined using a Turing-complete programming language, allowing the creation and execution of programs on the blockchain. Ethereum operates using accounts and their balances, which change through state transitions. The state represents the current balances of all accounts and additional data, and it is encoded and maintained by accounts in a separate data structure called a Merkle Patricia tree. Accounts on Ethereum are pseudonymous, providing a level of anonymity, and can be linked to one or more addresses. There are two types of accounts: externally owned accounts, controlled by individuals who possess private keys for transactions, and contract accounts, controlled by smart contract code. Contract accounts function as cyber-entities with their own balances and can be triggered by transactions from external accounts or other contracts. When triggered, the specified code in the contract is executed, which can generate further transactions. The presence of these smart contracts allows developers to use Ethereum as a versatile framework for creating DApps.

Figure 2.11 presents an overview of Ethereum blockchain, which consists of the following layers from bottom to top: peer, blockchain, smart contract, and token.

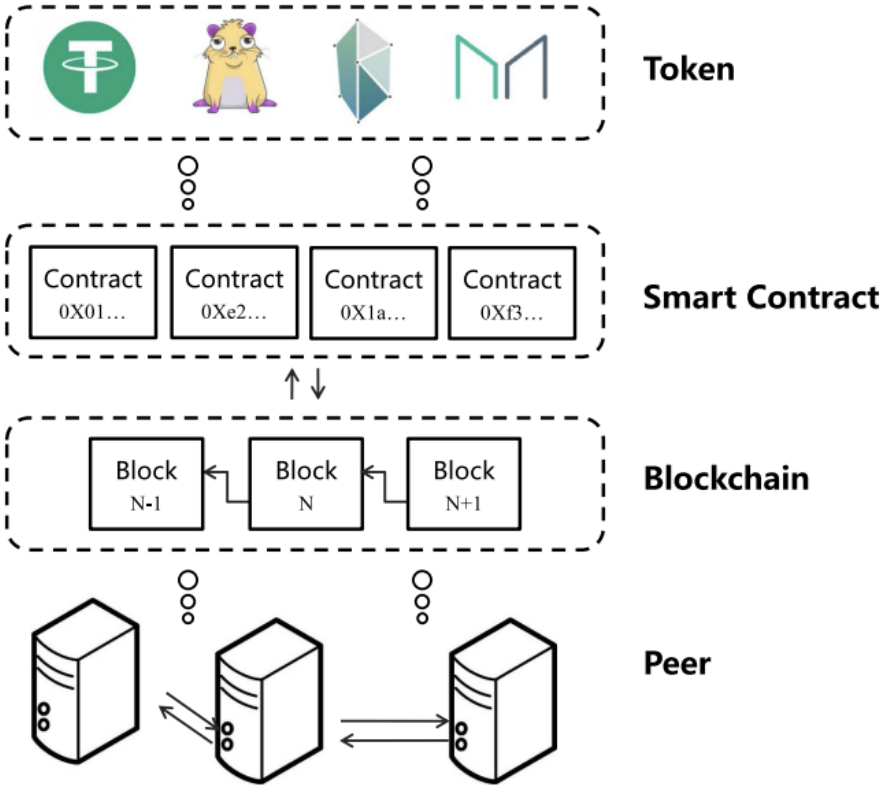


Figure 2.11: Ethereum Blockchain.

Ether is the cryptocurrency used within the Ethereum blockchain. It serves as fuel for operating distributed applications, making payments to other accounts or machines executing requested operations. Ether enables the execution of DApps, facilitates the use of smart contracts, allows the generation of tokens during Initial Coin Offerings (ICOs) for cryptocurrency-based funding, and supports standard peer-to-peer payments. This is why Ethereum is often referred to as "programmable money."

Ethereum Virtual Machine (EVM)

In Ethereum, the execution of a smart contract takes place within a specialized environment known as the EVM. The EVM interacts with the states stored as key-value pairs, based on the actions specified in the smart contract. Figure 2.12 illustrates the typical Ethereum transaction execution flow from Block N to EVM through Blockchain peer.

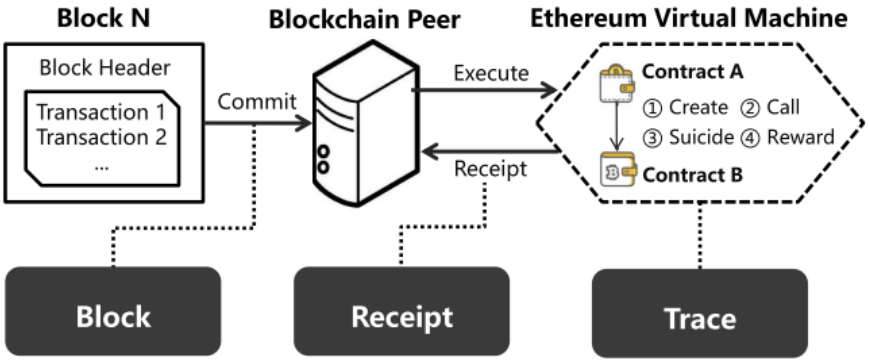


Figure 2.12: EVM role in blockchain.

During the execution of a contract, a miner utilizes "Gas" as a unit to measure the consumption of the contract. The contract user is charged based on the "GasUsed" and "GasPrice" parameters. If users offer a higher "GasPrice" to the miner, the contract will execute more quickly. Once the transactions or operations are completed, the EVM generates a hash value representing the state and records it in the blockchain.Consequently, it can be observed that smart contracts in Ethereum are not directly stored on the blockchain itself. Instead, they are essentially stored within the states that have been manipulated by the blockchain.

2.4.4 Canister

A smart contract is like a computer program that runs on a blockchain. On the Internet Computer, a smart contract takes the form of a canister, which combines the program itself with its associated data. Each canister is hosted on a specific subnet of the Internet Computer. One advantage of canisters is their ability to execute concurrently, meaning multiple canisters can run simultaneously. They can also communicate with each other within and across subnets by sending asynchronous messages, and this communication is non-blocking, allowing for virtually unlimited scalability.

Canisters are more powerful than smart contracts on other blockchains as they can:

- serve a user interface directly to any web browser.
- hold gigabytes of memory for a low fee.
- perform substantial amounts of computation at a low cost.
- pay for their own computation (reverse gas model).

The code of a canister is composed of a WebAssembly (Wasm) module, which is a type of module used in web development. The state of a canister includes the standard heap of the Wasm module, along with stable memory as shown in Figure 2.13. Stable memory is a special type of memory that plays a crucial role in the life cycle of a canister.

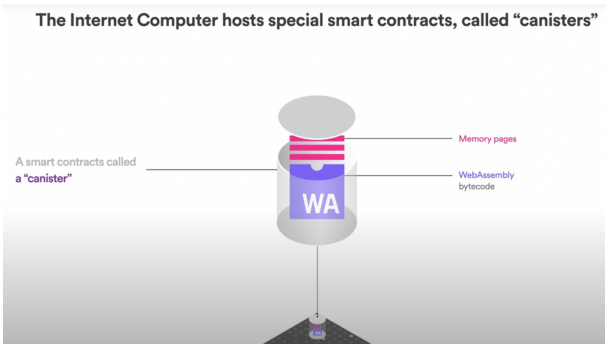


Figure 2.13: Overview of Canister.

Canisters on the Internet Computer have endpoints that can be accessed by other canisters as well as external parties like browsers or mobile apps. These endpoints come in two types: updates and queries. Updates are calls that have the ability to modify the state of a canister, while queries are limited to reading from the state. To simplify, think of updates as used for writing to the canister's state, and queries as used for reading from that state.

How Canister Work

Canisters on the Internet Computer operate in a manner similar to actors in the actor-based concurrency model. Each canister's code runs on a single thread and is completely isolated from other canisters. Canisters communicate with each other using asynchronous messaging. When a canister receives a message, it has the ability to modify its own state, send messages to other canisters, and even create new canisters. Unlike the traditional actor model, communication between canisters is two-way. Canister messages can be either requests or replies. When a request is sent, the Internet Computer (IC) keeps a record of a callback function to be executed when the receiving canister sends back a response. In cases where the IC determines that the receiving canister cannot respond, the system generates a response on behalf of the canister.

Another interesting aspect of the canister-based model is the interaction between message processing and canister trapping. While processing a request, a canister may send requests to other canisters and wait for some or all of the replies before generating a response to the original request. In the event that a canister encounters an issue or exception, it enters a trapped state, and its state is rolled back to the point immediately after it made the last outgoing call.

Canister Controllers

Canisters are supervised by controllers, which can be either users or other canisters. The management structure of canisters can take different forms: centralized (when controllers involve a centralized entity), decentralized (when the controller is a DAO), or even non-existent, in which case the canister is an immutable smart contract. Controllers are responsible for deploying and maintaining canisters on the IC, and they possess the authority to perform management operations on the canisters. Common operations include deploying a canister smart contract to the IC, as well as starting and stopping canisters. Controllers have the ability to modify canister parameters, such as adding or removing controllers and adjusting the freezing threshold.

Controllers can update the code running on canisters by submitting a new WebAssembly (Wasm) module to replace the older one. By default, updating the Wasm module erases the content of the Wasm memory, but the data stored in stable memory remains unchanged. The IC provides an upgrade mechanism that executes three actions atomically: serializing the Wasm memory of the canister and storing it in stable memory, installing the new Wasm code, and then deserializing the content from stable memory. However, a canister can ensure that data requiring persistence across upgrades is always stored in stable memory, simplifying the upgrade process significantly.

Canister behavior remains consistent.

Controllers are responsible for deploying and managing canister smart contracts. The level of decentralization of a controller can vary, ranging from an individual or a team of people to being managed by the Network Nervous System (NNS) [37] or another type of on-chain DAO. Controllers have the ability to modify the code of the canisters they control, making canister code mutable unlike traditional smart contracts on other blockchains. Additionally, controllers have complete control over the assets held by the canisters, such as ICP tokens or Bitcoins. This flexibility brings canisters closer to conventional software and enables them to be used in a wide range of applications where the logic of the software needs to be modified as required.

However, in critical applications like those used in Decentralized Finance (DeFi), centralized mutability can introduce risks. There is a possibility that a controller could maliciously modify a canister to steal assets. To address this concern, developers have several options available to them to verifiably decentralize the control of canister mutations.

2.5 Decentralized Applications (DApp)

In simpler terms, the blockchain can be understood as a distributed system that enables applications to operate on multiple nodes. Unlike centralized systems, blockchains

function based on decentralized consensus models, which means that applications on blockchains are DApps. DApps are a special type of software where the control over application execution is not held by a single entity. Previously, DApps were commonly associated with applications running on a Peer-to-Peer (P2P) network of computers rather than a single central computer. Examples of popular DApps include BitTorrent for file sharing, BitMessage for instant messaging, and Popcorn Time for video streaming.

By utilizing smart contracts, blockchains provide a flexible approach to computing, allowing the development of DApps for various application scenarios. For example, the Ethereum blockchain offers Turing-complete smart contracts that enable developers to create versatile programs. Consequently, as the popularity of blockchains continues to grow, a wide range of blockchain-based DApps have emerged and found applications in diverse domains. According to a recent report, the Ethereum DApps market, which is the largest blockchain-based DApps market, has reached a valuation in the billions of dollars as of January 2019 [38].

In other words, an ideal blockchain application or service should be operable without any human intervention, which forms a DAO. A DAO is an organization that is run through rules encoded as smart contracts running on the blockchain. Due to its autonomous and automatic nature, a DAO's cost and profit are shared by all participants by simply recording all activities into the blocks. In fact, Bitcoin, the most classic blockchain system, is an example of a DAO. According to the definition of DApps in [39], DApps are characterized by four properties as follow:

- *Open Source*: Due to the trusted nature of blockchain, DApps need to make their codes open source, so that audits from third parties become possible.

- *Internal Cryptocurrency Support*: Internal currency is the vehicle that runs the ecosystem for a particular DApps. With tokens, it is feasible for a DApps to quantify all credits and transactions among participants of the system, including content providers and consumers.

- *Decentralized Consensus*: The consensus among decentralized nodes is the foundation of transparency.

- *No Central Point of Failure*: A fully decentralized system should have no central point of failure since all components of the applications will be hosted and executed in the blockchain.

Table 2.2 shows top 10 DApps on State of the DApps and their numbers of related transactions from August 2017 to July 2018 respectively.

Table 2.2: Top 10 DApps in State of the DApps

Title	**Number of Transactions**
ForkDelta	9,575,884
IDEX	3,862,946
CryptoKitties	3,130,573
Bitcoinereum	1,451,752
OmiseGO	1,271,880
Storj	769,872
Ethereum Name Service	456,702
Status	407,946
PoWH 3D	327,353
Ether Goo	312,140

For each DApps, the website provides it's title, description, categories, submit time and smart contract addresses. It summaries transactions daily and generates sparklines of users, transactions and volumes. The website marks dapps with categories and badges (e.g. status), and ranks them by dau (the number of daily active unique addresses).

Chapter 3

Methodology

In this chapter, we present the methodology employed in the development of a simple banking application using Canisters on the Internet Computer. The aim of this project was to leverage the unique features and capabilities of Canisters to build a decentralized system for handling banking transactions and providing financial services to users.

3.1 Platform Selection

The decision to use the Internet Computer platform was driven by its decentralized nature, scalability, and robust security features. By utilizing the Technology Canister, which provides a secure and reliable environment for executing smart contracts and managing data, we were able to harness the power of the Internet Computer to build banking application.

1. **Internet Computer (IC)**

   The Internet Computer (IC) is a decentralized blockchain network that offers extensive computational power for executing canisters. This is made possible by a network of independent node providers who operate nodes across various data centers located around the world, ensuring scalability and distribution of resources.

2. **Canister**

   A canister is a unified entity that combines Wasm [40] program code and data storage. It is open for deployment by anyone on the Internet Computer. Canisters are stored and their code is executed in a replicated and fault-tolerant manner across multiple machines, known as nodes, within a subnet. In contrast to traditional blockchains, the Internet Computer allows for different mutability policies for smart contracts. A smart contract on the Internet Computer can adhere to various

mutability policies, such as complete immutability (where no changes can be made by anyone), unilateral mutability (allowing changes to be made solely by the dApp developer), or DAO mutability (permitting changes authorized by a decentralized autonomous organization).

3. **Canister Cycles**

   Canisters pay, using cycles, for the IC resources they consume. To this end, canisters need to be "topped up" with cycles. Cycles can be acquired with the ICP token, the IC's utility token. Buying cycles with ICP removes the ICP token from the supply and creates an amount of cycles with the corresponding value. One Trillion cycles can be acquired with ICP worth 1 XDR, where an XDR is a basket comprising major currencies and one XDR is roughly 1.3 USD as of Q3 2022.

4. **Controller**

   The controller of a canister refers to an individual, organization, or another canister that holds administrative privileges over that specific canister. Controllers are distinguished by their principals. For instance, a canister's controller possesses the authority to upgrade the canister's WebAssembly [40] code or delete the canister altogether.

5. **Principal**

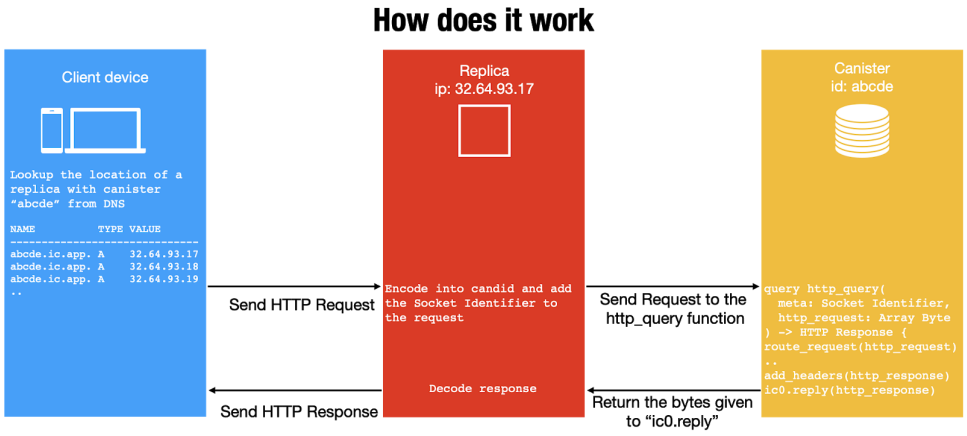
   A principal is an entity that can be authenticated by the Internet Computer. This is the same sense of the word principal as the Wikipedia [41] definition. Principals that interact with the Internet Computer do so using a certain identity.

6. **Replica**

   The replica is a collection of protocol components that are necessary for a node to participate in a subnet.

3.2 System Design

The development process of the web banking applications on the Internet Computer began by conducting a detailed analysis of the requirements and specifications. We then proceeded to design a comprehensive system architecture due to Figure 3.1 that encompassed all the essential components for a secure and efficient web banking experience. This included user registration, robust data storage mechanisms, and robust authentication protocols.



x

Figure 3.1: DBanking application architecture.

To ensure seamless integration and leverage the capabilities of the Internet Computer, the system was specifically designed to be compatible with the Technology Canister. By utilizing the features and functionalities provided by the Technology Canister, we were able to achieve a smooth and effective integration of the web banking applications into the Internet Computer ecosystem.

To gain a deeper understanding of the inner workings of Hypertext Transfer Protocol (HTTP) requests on the IC, let's explore the underlying process. Initially, when a client device initiates a request to a website, the website's domain name is resolved via Domain Name System (DNS), resulting in a list of Internet Protocol (IP) addresses belonging to replica nodes that store the target canister. The complete HTTP request is then forwarded to the appropriate replica, which proceeds to perform a query call to the canister. Subsequently, the canister responds with the HTTP response, which is relayed back to the replica. Finally, the replica transmits the response to the end user's device, where it is decoded and presented as the relevant website information.

3.3 Implementation

We implemented the DBank application by developing the frontend using Node.js and the backend with Motoko programming language. With the power of Canister technology, we deployed the application onto the Internet Computer, providing users with a decentralized banking experience. Through this implementation, we aimed to create a user-friendly interface and robust functionality, enabling individuals worldwide to access and utilize the benefits of DBank. Join us as we delve into the details of implementing DBank and discover the potential of decentralized finance.

Backend

I implemented this application using the Motoko programming language. Motoko is specifically designed for developing smart contracts and is well-suited for the programming model of the Internet Computer. It offers seamless integration with the unique features of the blockchain, allowing developers to leverage its capabilities effectively.

Motoko is a strongly typed language that follows the actor-based model, enabling easy communication and collaboration among actors. It also includes built-in support for orthogonal persistence and asynchronous message passing. The language provides several productivity and safety features, such as automatic memory management, generics, type inference, pattern matching, and support for both arbitrary- and fixed-precision arithmetic.

Additionally, Motoko transparently utilizes the Internet Computer's Candid interface definition language and wire format. This enables typed, high-level, and cross-language interoperability, making it easier to communicate with other components or systems within the Internet Computer ecosystem.

By utilizing Motoko as shown in Figure 3.2, I was able to take advantage of its powerful features and the seamless integration it offers with the Internet Computer, ensuring both productivity and safety in developing the application.

```
actor DBank {

  stable var currentValue : Float = 300;
  stable var startTime = Time.now();

  public func topUp(amount : Float) {…
  };
  public func withDraw(amount : Float) {…
  };
  public query func getBalance() : async Float {…
  };
  public func compound(){…
  };
}
```

Figure 3.2: Implementation Code.

In simple terms, the actor model is a conceptual framework for concurrent computing. It is a mathematical model that describes how individual actors behave when they receive

messages. When an actor receives a message, it can perform certain actions such as updating its internal state, sending messages to other actors, or even creating new actors. Essentially, the actor model allows actors to communicate and collaborate by exchanging messages and modifying their own state based on the messages they receive.

The code snippet represents an implementation of an actor called DBank in the context of a banking application. Let's break down the code and explain it in a humanized manner:

The DBank actor is designed to manage a user's bank account. It has a couple of stable variables: *currentValue* represents the current balance in the bank account, initialized to 300, and *startTime* stores the time when the account was created.

The actor provides several functions to interact with the bank account.

The *topUp* function as shown in Figure 3.3 allows the user to deposit money into their account. The amount parameter specifies the amount of money to be deposited. The function increases the *currentValue* by the deposited amount and prints the updated balance using the Debug.print function.

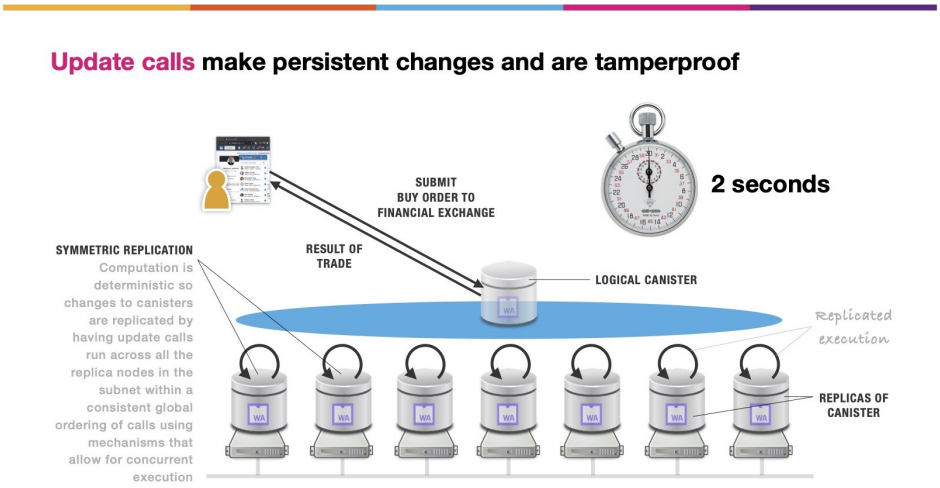


Figure 3.3: Update call.

The *withDraw* function enables the user to withdraw money from their account. The amount parameter specifies the amount to be withdrawn. The function checks if the withdrawal amount is less than or equal to the *currentValue*. If so, it subtracts the amount from the *currentValue* and prints the updated balance. If the withdrawal amount exceeds the available balance, it prints a message indicating that the user does not have enough money.

The *getBalance* function is a query function as shown in Figure 3.4 that allows other actors or clients to retrieve the current balance asynchronously. It returns the *currentValue* of the bank account.

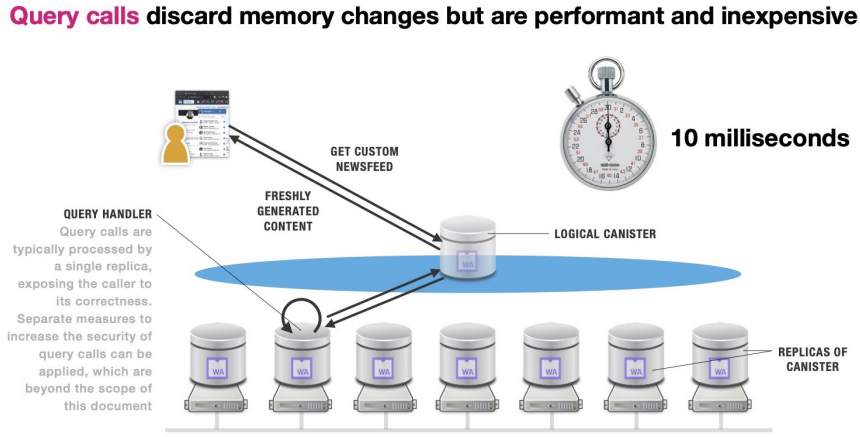


Figure 3.4: Query Call.

The compound function calculates compound interest on the bank account balance. It calculates the time difference between the current time and the *startTime* and converts it to seconds. Then, it applies a compound interest formula to update the *currentValue* by multiplying it with a factor of 1.01 raised to the power of the time difference in seconds. Finally, it updates the *startTime* to the current time.

By incorporating this code into your implementation subsection, you can explain how the DBank actor manages bank accounts, allowing users to deposit, withdraw, check their balance, and earn compound interest on their funds over time.

Frontend

To handle the frontend part of the application, I utilized the following code in Figure 3.5:

This code snippet is responsible for interacting with the frontend elements of the application. It first listens for the "load" event on the window, ensuring that the page has fully loaded before executing the code inside the event listener. Inside the event listener, the current balance is fetched from the dbank canister using the getBalance function. The retrieved balance is then displayed on the webpage by updating the corresponding HTML element. Furthermore, the code sets up an event listener for the form submission. When the form is submitted, the function inside the event listener is triggered. It retrieves the values entered in the input fields for the amount to be deposited and the amount to be withdrawn. The submit button is temporarily disabled to prevent multiple submissions while the transaction is being processed. If an input amount is provided, the topUp function from the dbank canister is called to perform the deposit. After the transaction is completed, the current balance is fetched again, and the updated balance is displayed on the webpage. Finally, the submit button's disabled attribute is removed, allowing subsequent transactions to be made. By utilizing this code, the frontend of the application interacts with the backend canister functions, enabling users to perform actions such as depositing funds and retrieving the current balance.

```
import { dbank } from "../../declarations/dbank";

window.addEventListener("load", async () => {
  const currentValue = await dbank.getBalance();
  document.getElementById("value").innerText =
    Math.round(currentValue * 100) / 100;
});
document.querySelector("form").addEventListener("submit", async (e) => {
  e.preventDefault();
  const button = e.target.querySelector("#submit-btn");

  const inputAmount = parseFloat(document.getElementById("input-amount").value);
  const outputAmount = parseFloat(
    document.getElementById("withdrawal-amount").value
  );

  button.setAttribute("disabled", true);

  if (document.getElementById("input-amount").value.length != 0) {
    await dbank.topUp(inputAmount);
  }
  const currentValue = await dbank.getBalance();
  document.getElementById("value").innerText =
    Math.round(currentValue * 100) / 100;

  button.removeAttribute("disabled");
```

Figure 3.5: Frontend Implementation Code.

Chapter 4

Experimental Results

During the experiments, I conducted transaction speed and scalability tests on the canister using Artillery. The main objective was to assess how quickly and efficiently the canister could process transactions. By subjecting the canister to varying levels of load, I simulated a high volume of transactions and measured the response time. The findings revealed that the canister exhibited impressive performance, maintaining a relatively fast response time even under heavy loads. This is a significant advantage as it demonstrates the canister's ability to handle a large number of transactions without compromising its efficiency, ensuring a smooth user experience.

4.1 Transaction Speed

The Internet Computer distinguishes between two types of smart contract function executions: update calls and query calls. Update calls are requests made to modify the state of canister smart contracts. These calls initiate changes and updates within the smart contract system. On the other hand, query calls, which constitute a significant majority (over 90%) of the Internet Computer's traffic, are read-only requests. These calls are designed for retrieving information and do not involve any changes to the state of the smart contract. Notably, query calls are exceptionally efficient, with execution times as short as 1.5 milliseconds on the Internet Computer platform.

After conducting load testing using Artillery,As Shown in Figure 4.1 ,to evaluate the performance of the canister, we compared two sets of load testing results. The analysis of these results reveals the following observations:

For the first set of results (Query):

- The maximum response time recorded was 291 milliseconds.
- The 99th percentile response time, representing the response time for the vast majority of requests, was 23.8 milliseconds.

In the second set of results (Update):

- The maximum response time reached 304 milliseconds.
- The 99th percentile response time measured at 22.9 milliseconds.

By comparing these metrics, we can see that the first set of results exhibits a slightly higher maximum response time (291 ms) compared to the second set (304 ms). However, the 99th percentile response time for the second set (22.9 ms) is lower than that of the first set (23.8 ms).

Considering the 99th percentile response time as a measure of overall performance, we can conclude that the second set of results demonstrates slightly faster performance.

It is important to note that the differences in response time between the two sets are relatively small. The choice between the sets may depend on other factors or specific requirements of the tested application or system.

```yaml
config:
  target: "http://127.0.0.1:8001"
  phases:
    - duration: 100
      arrivalRate: 50

scenarios:
  - name: "Canister Test Query"
    flow:
      - post:
          url: "/?canisterId=ryjl3-tyaaa-aaaaa-aaaba-cai"
          json:
            functionName: "getBalance"
    count: 200
    continueOnError: true
  - name: "Canister Test Update"
    flow:
      - post:
          url: "/?canisterId=ryjl3-tyaaa-aaaaa-aaaba-cai"
          json:
            functionName: "topUp"
            amount: 100
    count: 200
    continueOnError: true
```

Figure 4.1: Canister Performance Test

As we The following example generates 50 virtual users every second for 1.66 minutes. Both result sets have a high number of successful responses (‘http.codes.200: 5000‘) and completed virtual users (v̀users.completed: 5000‘). The request rate is the same (h̀ttp.request_rate: 50/sec‘), indicating a consistent load applied in both tests.

4.2 Scalability Test

The provided test configuration ,As shown in Figure 4.2, is designed to simulate a load on a target server running locally at "http://127.0.0.1:8001". The test consists of two scenarios: "Canister Test Query" and "Canister Test Update". During the test, 500 virtual users are created per second for a duration of 10 seconds, generating a total of 5000 requests. The "Canister Test Query" scenario involves sending a POST request to the specified URL with a JSON payload containing the function name "getBalance". Similarly, the "Canister Test Update" scenario sends a POST request with a different function name "topUp" and an amount of 100. Both scenarios are repeated 200 times, and errors encountered during the test are allowed to continue without halting the execution. Based on the test configuration, it indicates that the canister is capable of handling the load generated by 500 users per second for the given duration.

As for Table 4.1, the scalability testing results for the Canister application are as follows:

Table 4.1: Summary of Metrics for Scalability Test

Metric	**Value**
Total HTTP requests	5000
Successful HTTP responses (status code 200)	5000
Request rate	506 requests per second
Minimum response time	7 milliseconds
Maximum response time	4118 milliseconds
Median response time	3011.6 milliseconds
95th percentile response time	3984.7 milliseconds
99th percentile response time	3984.7 milliseconds
Total virtual users created	5000
Virtual users created for Canister Test Query	2486
Virtual users created for Canister Test Update	2514
Failed virtual users	0
Minimum session length	8 seconds
Maximum session length	4210 seconds
Median session length	3011.6 seconds
95th percentile session length	3984.7 seconds
99th percentile session length	3984.7 seconds

```yaml
config:
  target: "http://127.0.0.1:8001"
  phases:
    - duration: 10
      arrivalRate: 500

scenarios:
  - name: "Canister Test Query"
    flow:
      - post:
          url: "/?canisterId=ryjl3-tyaaa-aaaaa-aaaba-cai"
          json:
            functionName: "getBalance"
    count: 200
    continueOnError: true
  - name: "Canister Test Update"
    flow:
      - post:
          url: "/?canisterId=ryjl3-tyaaa-aaaaa-aaaba-cai"
          json:
            functionName: "topUp"
            amount: 100
    count: 200
    continueOnError: true
```

Figure 4.2: Canister Scalability Test

These results provide insights into the scalability of the Canister application. By analyzing the report, it is possible to assess how the system performs under load and identify any potential bottlenecks or issues related to scalability.

Chapter 5

Conclusion

In conclusion, this project has made significant contributions to the field of blockchain and decentralized applications. By improving transaction speed and developing a secure smart contract execution framework, we have addressed key challenges in blockchain technology. However, there is still much to explore in terms of scalability, privacy, and interoperability. The future works outlined above provide valuable directions for researchers and developers to further advance the capabilities and applicability of blockchain technology.

5.1 Main Contributions

In this thesis, we have accomplished some significant achievements in the field of web application development using canisters and blockchain technology. Let me break down our contributions for you:

1. Delving into the world of blockchain architecture: We embarked on a comprehensive exploration of the architecture of blockchain, uncovering its decentralized nature and the fundamental principles that govern its operation. This understanding forms the bedrock for utilizing canisters as a decentralized approach to web application development.

2. Solving the double spending problem: We tackled a critical issue in blockchain networks—double spending—and delved into innovative solutions to mitigate this problem. Through our analysis and evaluation of these solutions, we have added to the existing knowledge about the security and integrity of blockchain-based systems.

3. Discovering the diverse applications of blockchain: We went on a journey to explore the diverse applications of blockchain technology across various industries, including climate, supply chain management, and healthcare. Our investigation shed light on how blockchain can bring transformative changes to these sectors, providing valuable insights for researchers, businesses, and decision-makers.

4. In-depth research on canisters: We dedicated extensive effort to conducting in-depth research and practical analysis of canisters. We focused on their architecture, programming model, and deployment procedures to gain a comprehensive understanding of how to utilize them in web application development. Our findings contribute to the technical knowledge required for effectively leveraging canisters.

5. A captivating case study: To demonstrate the real-world potential of canisters, we developed a fully functional web application. This case study not only showcased the advantages of using canisters but also highlighted the complexities involved. We tackled challenges such as data storage, user authentication, and transaction management, providing valuable insights for developers and businesses.

Overall, our research expands the existing knowledge in decentralized computing and emphasizes the transformative power of canisters in web application development. Our insights and findings aim to provide practical guidance and inspiration for developers, businesses, and end-users who are eager to harness the benefits of blockchain technology in their innovative web-based solutions.

5.2 Future Work

While this project has achieved significant progress in the field of blockchain and decentralized applications, there are several areas that warrant further exploration and improvement. The following future works outline potential directions to enhance the update call performance and facilitate the deployment of web applications on canisters and the Internet Computer:

1. Optimization of Update Call Speed

   One crucial area for future research is the optimization of update call speed in canisters. Despite the advancements made in smart contract execution, there is still room for improving the efficiency of update calls. Exploring novel techniques, such as parallel processing, caching mechanisms, or optimizing the communication protocols between canisters and clients, can help reduce the latency and improve the responsiveness of update calls. By enhancing the update call speed, we can enable faster and more interactive decentralized applications.

2. Streamlining Web Application Deployment on Canisters

   Another important direction for future development is streamlining the deployment process of web applications on canisters and the Internet Computer. Simplifying the steps required to deploy web applications, including packaging dependencies, configuring network access, and managing application state, can lower the barrier to entry for developers and encourage broader adoption of canister-based applications. Investigating automated deployment tools, integrating with popular web frameworks, and providing comprehensive documentation can facilitate the seamless deployment of web applications on the Internet Computer.

Appendix

Appendix A

Lists

List of Abbreviations

DDoS	Distributed Denial of Service
ARP	Address Resolution Protocol
ECC	Elliptic Curve Cryptography
MHT	Merkle Hash Tree
HB	Hash of B
HCD	Hash of CD
ECDH	Elliptic Curve Diffie-Hellman
BGP	Border Gateway Protocol
DAO	Decentralized Autonomous Organization
Wallet	Wallet Attack
Double Spending	Double Spending Attack
Spam	Spam Attack
Selfish Mining	Selfish Mining Attack
EVM	Ethereum Virtual Machine
MIT	Massachusetts Institute of Technology
SVM	Symbolic Virtual Machine
CHC	Constrained Horn Clauses
SMT	Satisfiability Modulo Theory
LLVM	Low Level Virtual Machine
CH	Cluster Head
HMM	Hidden Markov Model

PoS	Proof-of Stake
BCTLF	Blockchain Transportation and Logistics Framework
IoT	Internet of Things
IoV	Internet of Vehicles
EV	Electric Vehicle
REDD+	Reducing Emissions from Deforestation and Forest Degradation
EMRs	Management of Electronic Medical Records
EHRs	Electronic Health Records
PHRs	Personal Health Records
GDPR	General Data Protection Regulation
GHN	Gem Health Network
BFT	Byzantine Fault Tolerant
eGov-DAO	Electronic Government Decentralized Autonomous Organization
CGC	Cyber Grand Challenge
DARPA	Defense Advanced Research Projects Agency
SMT-LIB	Satisfiability Modulo Theories
SODA	Smart Contract Online Detection Freamwork against Attacks
DApps	Decentralized Applications
ICOs	Initial Coin Offerings
IC	Internet Computer
Wasm	WebAssembly
NNS	Network Nervous System
DeFi	Decentralized Finance
P2P	Peer-to-Peer
HTTP	Hypertext Transfer Protocol
DNS	Domain Name System
IP	Internet Protocol

List of Figures

List of Tables

Bibliography

[1] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology — CRYPTO '87* (C. Pomerance, ed.), (Berlin, Heidelberg), pp. 369–378, Springer Berlin Heidelberg, 1988.

[2] Protocol Labs, "IPFS powers the distributed web." https://ipfs.io/.

[3] M. Bell, "Proof-of-stake bitcoin sidechains." https://gist.github.com/mappum/da11e37f4e90891642a52621594d03f6, June 2021.

[4] R. Jabbar *et al.*, "Blockchain technology for intelligent transportation systems: A systematic literature review," *IEEE Access*, vol. 10, pp. 20995–21031, 2022.

[5] T. L. Nguyen, "Title of the paper," in *2018 Portland International Conference on Management of Engineering and Technology (PICMET)*, pp. 1–6, IEEE, Aug 2018.

[6] R. Kumar, N. Marchang, and R. Tripathi, "Title of the paper," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 1–5, IEEE, Jan 7 2020.

[7] S. Angraal, H. Krumholz, and W. Schulz, "Blockchain technology applications in health care," *Circulation: Cardiovascular Quality and Outcomes*, vol. 10, p. e003800, 2017.

[8] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *Proceedings of the 2016 IEEE 18th International Conference on E-Health Networking, Applications and Services (Healthcom)*, (Munich, Germany), pp. 520–522, September 2016.

[9] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Proceedings of the 2016 2nd International Conference on Open and Big Data (OBD)*, (Vienna, Austria), pp. 25–30, August 2016.

[10] M. Engelhardt, "Hitching healthcare to the chain: An introduction to blockchain technology in the healthcare sector," *Technological Innovation Management Review*, vol. 7, pp. 22–34, 2017.

[11] V. Y. Kemmoe, W. Stone, J. Kim, D. Kim, and J. Son, "Recent advances in smart contracts: A technical overview and state of the art," *IEEE Access*, vol. 8, pp. 117782–117801, 2020.

[12] "Ethereum-white paper." Accessed: Jun. 5, 2019. https://github.com/ethereum/wiki/wiki/White-Paper.

[13] "Eosio-cleos/cleos-set." Accessed: Jul. 10, 2019. https://developers.eos.io/eosio-cleos/v1.2.0/reference#cleos-set.

[14] "Neocontract white paper." Accessed: Oct. 24, 2019. https://docs.neo.org/docs/en-us/basic/technology/neocontract.html.

[15] "Eosio api." Accessed: Oct. 24, 2019. [Online]. Available: https://eosio.github.io/eosio.cdt/latest/modules.

[16] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM (JACM)*, vol. 27, pp. 228–234, April 1980.

[17] G.-T. Nguyen and K. Kim, "A survey about consensus algorithms used in blockchain," *Journal of Information Processing Systems*, vol. 14, no. 1, pp. 101–128, 2018.

[18] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *Proceedings of the 31st International Symposium on Distributed Computing (DISC)* (A. W. Richa, ed.), vol. 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 39:1–39:16, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[19] Y. Wu, P. Song, and F. Wang, "Hybrid consensus algorithm optimization: A mathematical method based on pos and pbft and its application in blockchain," *Mathematical Problems in Engineering*, vol. 2020, pp. 1–13, Apr. 2020.

[20] "Solidity documentation." https://solidity.readthedocs.io.

[21] E. Dmitry and P. Roschin, "The all-pervasiveness of the blockchain technology," *Procedia Comput. Sci.*, vol. 123, pp. 116–121, 2018.

[22] W. Shuai, Y. Yuan, X. Wang, J. Li, R. Qin, and F.-Y. Wang, "An overview of smart contract: Architecture, applications, and future trends," in *Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV)*, (Changshu, China), pp. 26–30 June, 2018.

[23] H. Everett, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, and B. Moore, "Kevm: A complete formal semantics of the ethereum virtual machine," in *Proceedings of the 2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, (Oxford, UK), pp. 9–12 July, 2018.

[24] K. Ahmed, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, (San Jose, CA, USA), pp. 22–26 May, 2016.

[25] B. Imran, *Mastering Blockchain: Distributed Ledger Technology, Decentralization, and Smart Contracts Explained*. Birmingham, UK: Packt Publishing Ltd., 2018.

[26] S. Khaled, M. Rehman, N. Nizamuddin, and A. Al-Fuqaha, "Blockchain for ai: Review and open research challenges," *IEEE Access*, vol. 7, pp. 10127–10149, 2019.

[27] S. Christian, B. Waltl, C. Sillaber, and B. Waltl, "Life cycle of smart contracts in blockchain ecosystems," *Datenschutz Und Datensicherheit-DuD*, vol. 41, pp. 497–500, 2017.

[28] B. Massimo and L. Pompianu, "An empirical analysis of smart contracts: Platforms, applications, and design patterns," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, (Sliema, Malta), pp. 3–7 April, 2017.

[29] C. Pierluigi, "Beyond bitcoin: An early overview on smart contracts," *Int. J. Law Inf. Technol.*, vol. 25, pp. 179–195, 2017.

[30] O. Steve, "Cryptocurrencies, smart contracts, and artificial intelligence," *AI Matters*, vol. 1, pp. 19–21, 2014.

[31] N. Buciek and P. Sandner, "The blockchain technology in the media sector," *Media Trust in a Digital World: Communication at Crossroads*, pp. 207–218, 2019.

[32] N. Diallo, W. Shi, L. Xu, Z. Gao, L. Chen, Y. Lu, N. Shah, L. Carranco, T.-C. Le, A. B. Surez, *et al.*, "egov-dao: A better government using blockchain based decentralized autonomous organization," in *2018 International Conference on eDemocracy & eGovernment (ICEDEG)*, pp. 166–171, IEEE, 2018.

[33] D. Mark, "Does contracting out increase the efficiency of government programs? evidence from medicaid hmos," *Journal of Public Economics*, vol. 88, pp. 2549–2572, 2004.

[34] M. Mossberg, F. Manzano, E. Hennenfent, A. Groce, G. Grieco, J. Feist, T. Brunson, and A. Dinaburg, "Manticore: A user-friendly symbolic execution framework for binaries and smart contracts," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1186–1189, IEEE, 2019.

[35] T. Chen, R. Cao, T. Li, X. Luo, G. Gu, Y. Zhang, Z. Liao, H. Zhu, G. Chen, Z. He, *et al.*, "Soda: A generic online detection framework for smart contracts.," in *NDSS*, 2020.

[36] M. Bez, G. Fornari, and T. Vardanega, "The scalability challenge of ethereum: An initial quantitative analysis," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 167–176, IEEE, 2019.

[37] "Internet computer - network nervous system." https://internetcomputer.org/nns. Accessed on May 25, 2023.

[38] S. A. Abdulhakeem, Q. Hu, *et al.*, "Powered by blockchain technology, defi (decentralized finance) strives to increase financial inclusion of the unbanked by reshaping the world financial system," *Modern Economy*, vol. 12, no. 01, p. 1, 2021.

[39] S. Raval, *Decentralized applications: harnessing Bitcoin's blockchain technology.* " O'Reilly Media, Inc.", 2016.

[40] WebAssembly, "Webassembly," Accessed 2023. https://webassembly.org/.

[41] Microsoft, "Understanding security principals," Accessed 2023. https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-principals.