National College of Ireland

Upgradeable Smart Contracts design patterns for Dapps Architectures

MSc Research Project
MSc in Cloud Computing

Joan Carlo Lopez Marin
Student ID: 20232535

School of Computing
National College of Ireland

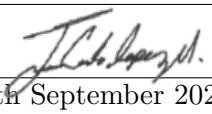
Supervisor: Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Joan Carlo Lopez Marin
Student ID:	20232535
Programme:	MSc in Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Shivani Jaswal
Submission Due Date:	15/08/2022
Project Title:	Upgradeable Smart Contracts design patterns for Dapps Architectures
Word Count:	5998
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	19th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☑

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Upgradeable Smart Contracts design patterns for Dapps Architectures

Joan Carlo Lopez Marin
20232535

Abstract

Current classifications for smart contract design patterns see the smart contract as an isolated entity running in a block-chain. Most of them were created when the gas consumption was not a big deal. The majority of the actual classifications do not take into account how much gas the patterns proposed spend. The aim of this work is to develop guidelines to help the developer to take better architectural decisions during the planning of the distributed backend application, which is usually composed of multiple smart contracts and many other off-chain technologies that work together to handle the block-chain limitations. Some approaches and design patterns are selected from different sources, and all patterns are evaluated to measure efficiency in terms of gas consumption and trustworthiness from the user perspective in terms of using centralized or decentralized technologies to provide the developer with a better understanding of the implications of its implementation. The selected patterns have the core philosophy that smart contracts are a small piece of a dapp, which can be made up of multiple technologies in-chain and off-chain. A framework is proposed that has the upgradability as a central concept and the most relevant patterns to accomplish with the most common software application requirements.

1 Introduction

A staggering number of blockchain systems have been developed since Nakamoto released the paper for Bitcoin in 2008 [1]. All industries are embracing these new platforms, from gaming to enterprise businesses, but since it is still relatively recent, the protocols, use cases, and business norms are still developing.

Smart Contracts, which are automatically executed contracts implemented on a blockchain [2], are one of the most relevant and well-liked enhancements to blockchain platforms. The first and still most popular blockchain that combined these two ideas was Ethereum [3, 2]. The processing of digital currency is the most frequent application for smart contracts, thus it is crucial to pay close attention to the development process.

One of the main reasons why smart contracts are now used in blockchains is that they offer a fascinating way to improve the security and transparency of the data managed; as a result of their growing use, a proposal known as Dapps, which are applications that run in a blockchain at least one decentralized component, was introduced[4].

Like in all new technology, with smart contracts developers were confronted with some new challenges while developing Dapps. To address these issues, multiple new frameworks, and libraries were created, along with new design patterns. The design

patterns are similar to recipes and often include a problem description with the solution [5].

Several writers have previously detailed a number of design patterns used by diverse blockchain systems like [6], [7], [8], [9], [10]. The problem with the patterns presented is the analysis of smart contracts separately. Nevertheless, with the increment use of Dapps, multiple smart contracts may serve as the backend of a client application [11]. A fascinating approach is the partial decentralization of an application or its distribution across multiple decentralized systems; to illustrate, an application can manage a token in blockchain, but the data process can be handled by a centralized system or all the persistence in a peer-to-peer distributed storage network system such as IPFS. In light of the fact that the smart contract may interact with multiple contracts conforming the backend of the application among other systems.

Even when the actual presented patterns are focused on dealing with the smart contracts behavior separately, **may these patterns be employed as part of a distributed backend application?**. This study aims to assist developers in making the optimal architectural choices based on the needs of the software solution.

The sections of this research are listed in this way, from sections 2.1 to 2.3 a little background of the technologies involved, then in the section 2.4 Some proposed classifications for design patterns are discussed, then in the section 2.5 the weaknesses of the blockchain and smart contracts are listed, then in the section 3 the characteristics and criteria used to select the patterns to evaluate is described, in section 4 are listed the steps to evaluate the patterns, section 5 has the technologies used, in the section 6 the patterns are evaluated, and section 7 the conclusion and future work are listed.

2 Related Work

As the blockchain, smart contracts, and Dapps are relatively new technologies, this section describes a small introduction of each of them as well as the relationship with the design patterns and the actual classifications proposed by other authors.

2.1 Blockchain

During the early years of the 1990s, the authors in [12] were the first to introduce the blockchain idea. In a new age in which changing digital data is simple, they found a method to protect and defend digital information. Using a distributed electronic ledger containing time-stamped data units referred as blocks, they provided a mechanism to verify when a document was generated or edited. Each block carries a hash, and this hash also appears in the following block. This establishes a chain, and if the user wants to edit one block, he must modify the whole chain. Additionally, this chain is allocated in distinct, interconnected nodes. This safeguards the data stored inside the chain against tampering or fraud.

Four years after the patent for this innovation expired in 2004 due to nonpayment of maintenance costs, the paper [1] was released in 2008, using the blockchain system to build the Bitcoin. In this document, the author was simultaneously presenting two radically new concepts that had never been combined before: Bitcoin, a decentralized p2p digital cash system that maintains value out of a bank or central entity, and the proof-of-work built blockchain system that allows agreement on the order of transactions.This article brought blockchain technology to the attention of the global community, and as

time passed, many additional protocols and features were developed to test blockchain capabilities, the majority of these new applications serving specialized needs [13].

2.2 Smart Contracts

Implementation of smart contracts is one of the most significant enhancements to blockchain technology. In 1997, the first mention of a smart contract appeared in [14], the article describes the algorithm used to transfer certain rights. The main premise has evolved throughout the course of successive publications [15] and [16]. Even though the author outlined how these contracts might be crucial to the future collaboration of humankind, no framework is suggested to execute them.

Now, the principle of smart contracts is applied on a blockchain, which was first introduced on Ethereum, a deterministic Turing-based virtual system. These are basically the conditions of an agreement between numerous parties that are specified in code lines. This code is implemented on a decentralized blockchain system that enables parties to make anonymous and secure transactions while they do not need a central authority to perform or verify the contract because it is self-executed. These contracts execute transactions that are traceable, transparent, and immutable. To be a smart contract needs to have the following characteristics:

1. Deterministic
2. Terminable
3. Immutable

In the same article, the author introduced several significant topics, but for the purposes of this research, we will just discuss Gas. "Gas" in Ethereum is the fundamental network cost unit, which is the amount you must pay for the processing power required to complete a transaction. Essentially is a variable transaction fee that must be paid in the native token of the blockchain, in Ethereum is paid in Ethers. Other blockchain networks employ the same principle but with a different name [17].

A smart-contracts is software that is deployed in a blockchain, it can be written in different languages depending of blockchain, some of them support multiple languages, in the case of Ethereum, the Ethereum Virtual Machine supports Solidity and Vyper [18], Solana support C++ and Rust [19] and so on. Solidity is the most popular language and it was created by the community as a contract focused language.

As it is extremely difficult for the user to predict how much gas will be utilized during an operation, the user sets a gas limit before initiating an operation. This is the maximum amount of gas the user is willing to spend on this operation. If the amount of gas needed by the transaction exceeds the limit, the process is terminated, the EVM rolls back each change, and all the gas spent is lost. If a greater gas limit is established, just the necessary amount of gas will be used. It is bad practice to set a really high max gas value since if an error or bug occurs during the execution of the contract, all the gas might be lost.

As the environment where the smart contracts are deployed is different than other applications, the paradigm for designing them is different, and the general characteristics of smart contracts deployed in Ethereum are detailed in [20]. According to the article, on average, the contracts contain around fifty-seven lines and five functions. Another fascinating data presented by the author is the number of ancestors of the contract,

which indicates that the average number of direct or indirect predecessors a contract has is two, while contracts with more ancestors have thirty.

The capability and features to execute transactions employing smart contracts in a blockchain are the core of Dapps, a new kind of application that utilizes this technology as a backend and storage.

2.3 DApps

In 2004 a new term was coined to make a reference to the evolution of the web, "web2.0" has been used to characterize the progression of web applications that include responsive design, interactivity, and user-generated content. This is not a technical standard but rather a concept used to characterize the web's primary characteristics. Web 3.0 is the concept introduced in [11] to describe the evolution of web applications that introduces the implementation of decentralized peer-to-peer protocols. The implementation of smart contracts on a blockchain introduces Dapps, an innovative new category of software applications. Dapps refer to distributed applications. A DApp is, in the simplest words, a web app built on open, decentralized, peer-to-peer infrastructure services.

To define an application as a dapp, it needs to have the following two components:

1. Smart Contract deployed in a blockchain
2. A front-end user interface

To deal with the limitations of the blockchain, some dapps also implement other decentralized components:

1. Decentralized peer-to-peer storage service
2. Decentralized peer-to-peer message service

2.4 Design Patterns

It is common for software developers to use the same paradigm and related technology to create software solutions that encounter similar situations. Consequently, the meaning of design patterns was introduced in the early 1990s. A design pattern describes a problem and its solutions. It resembles a recipe [5].This concept is used in blockchain technologies to deal with the limitations and issues.

One characteristic that differentiates the patterns in blockchain from other technologies is that most of the patterns that already exist to face the limitations of the blockchain can be implemented in one Dapp. In contrast with parallel design patterns implemented in parallel computing where the frameworks are created implementing just one pattern to solve the parallelization issues [5], the smart contracts are a small piece of a puzzle, and multiple contracts can create the backend of a dapp, each piece can be designed with different patterns to solve the specific problems.

Design patterns are not a Nobel idea. Even in blockchain, there are some authors that have already published different classifications for different objectives, grouping them into different taxonomies. One of these classifications is described in [6] where the author describes four categories.

1. Creational Pattern

2. Structural Pattern

3. Inter Behavioral Pattern

4. Intra-behavioral pattern

The categorization proposed by the author in general terms is grouped by functionality but keeping all the functionalities inside the blockchain, which could be expensive because they are not taking into account the gas consumption in blockchains like Ethereum.

Another proposed categorization is described in [9] where the author takes upgradability as a core characteristic, basically, this author proposes the separation of the data contract from the logic and suggests a proxy contract that can be used as a proxy to deal with the immutability. It works as follows: The first contract is the proxy contract. This contract is the main endpoint that gets requests from users, applications, and other contracts. In this contract, the address of the second contract is stored in a variable that can be updated. Then the second contract is deployed. This contract contains all the logic or data in the case of the data segregation pattern. If the contract with the process needs to be updated, then it is not necessary to update all the clients that are consuming that contract. The new address for the contract that contains the updated logic can be changed in the variable of the first contract. The main concern about this categorization is that everything is also managed inside the blockchain.

In contrast to the categorizations described above, the author in [10] propose off-chain technologies to deal with the limitations of the blockchain. The author describes a chess application created with all the data and logic inside the blockchain. In general, the application works as follows: First, all moves are verified for a smart contract. After a legal move is executed, the persistent data is updated, and the contract verifies the end game rules. If the game is finished, the contract sends the tokens to the winner. Performing all the functions listed and storing all the data inside the chain could be a costly approach. Each move generates data and executes functions; this performs multiple operations. All operations have costs in blockchain, and there is no way to perform computations on private data without revealing the content. The solution proposed by the authors is the management of the information and processes outside of the chain.

Most of the approaches listed above were created to deal with the blockchain issues, the approach described in [8], uses the patterns listed by the other authors but focuses on the patterns that help with gas consumption.

2.5 weakness of blockchain and smart contracts

The classifications and patterns mentioned above were created to solve the limitations of blockchain and smart contracts. We can describe these limitations as follow:

1. Upgradability: One of the most important characteristics of blockchain is immutability, which means once a transaction is executed, it cannot be changed because all transactions, executions, and smart contracts are stored as a block in the chain. In the case of transactions, it is an advantage, but it becomes an issue when the developer is working on a dapp backend(smart contracts) because all deployments generate costs [21], and in most cases, the software needs to be updated because of new business rules, bugs, or many other situations. In blockchain, the only way to update a contract is to deploy a new one because the contracts are stored as blocks in the chain. They can not be modified or deleted, so the update process for smart

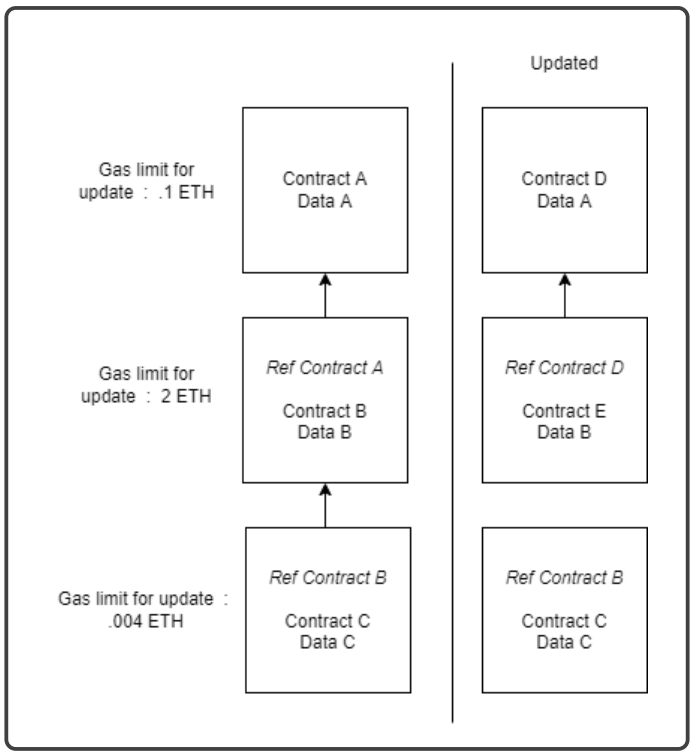


Figure 1: Gas price during update contract process

contracts needs a manual migration of all the information from the old contract to the new one. This migration generates costs as well.

At the moment of the migration the developers need to pay special attention to the following points:

(a) Gas Limit: Updates may be extensive depending on the size of processes required to accomplish the migration, such as deploying a new contract, migrating data, and migrating references.

(b) Inter contract dependencies : In the moment of the compilation of the contract, all the contracts imported for this contract are compiled in the contract as well, if one contract is updated, the update creates a domino effect, where all the contracts that are importing that contract have to be updated as well.

In the figure 1 are 3 contracts that are deployed in a blockchain like Ethereum, all contracts contain different data and for that reason the gas price for update each one is different. In the left side are represented the smart contracts before the updating process and in the right side are the contracts after updating process. All the data from the contracts A and B needs to be migrated to the contracts D and E, and the total cost of the update process is 2.1ETH. The contract C was not updated and still imports the old version of the contract B that is not maintained any more.

2. Processing and Data Management: All processes and data management in a blockchain need to be validated and executed in a transaction block, all processes and validations are performed by a miner, and the service of the miner has a cost that

depends on the size and number of the operations, this process could be expensive and can be slow. In Ethereum the transaction limit is 15 transactions/second [2], this could be a problem at the moment to scale and in a public blockchain the privacy is not granted because everything can be visible.

3. Randomness: Consensus is an essential characteristic of the blockchain for all the miners involved in the network, it means they have to obtain the same result when evaluating some transactions. For this reason, the blockchains that can execute smart contracts are created as deterministic Turing Machine [2] [22], and the randomness could be an issue in the network consensus.

4. Encapsulated Network: All the processes and transactions have to be performed, validated and replicated by miners that are in the network, and the contracts can not get information outside of the network.

3 Methodology

All of the technologies involved in the blockchain and dapps come with disadvantages. In the case of smart contract to deal with their disadvantages, some writers propose the use of design patterns to help the developers during design, development, and deployment to create better systems.

The patterns proposed help with the disadvantages of the blockchain and smart contracts. Each author selects the patterns with different characteristics, but the problem is that they create a categorization based on the principle that the smart contract is an isolated entity. So at the moment of the architectural design is a complex task to select the better options regarding design patterns to be implemented in a Dapp. The dapps is still an application with at least three layers that must be covered The layers are:

1. Persistence: Is responsible to storing the data, files and any information.

2. Logic: Is responsible to manage the data processing and transactions.

3. Client: Is responsible to provide the user interface.

Persistence and logic are the layers managed by the blockchain. For this reason, we will focus on covering only those layers in this research.

For each smart contract pattern selected the developer needs to take into account 3 concepts that becomes from the nature of the blockchain.

1. Gas Consumption :All operations, deployments, and transactions in blockchain generate costs. All of the characteristics listed above must be implemented in Dapps, considering the gas consumption.

2. Trust: By the nature of the blockchain as a distributed ledger with public transactions validated by multiple nodes, the user can be aware about what is happening during the execution of a smart contract, and this provides confidence from the user perspective. To provide this trust, the process has to be running inside a blockchain and this can be expensive in large operations.

3. Privacy: In the contrary of the trust, in some cases there is information that needs to be private. Blockhcain can not guarantee full privacy because there is no way to perform computations on private data without revealing the content. To provide this privacy the process needs to be performed outside of the blockchain, this provide privacy and is cheaper but the user can not be aware about what exactly is happening during the execution, the trust from the user perspective is impacted.

We identified the primary characteristics that centralized applications have and a decentralized application needs to accomplish with the most common software demands. We use these characteristics as a Categorization.

1. Data/File Storage: As shown in the figure 2 the main function of the patterns in this category is to deal with the persistence layer, in means : files and data. This category is divided in two more categories:
   (a) On-Chain data management: Manage all the information inside the blockchain could be very expensive but in some applications is a necessity to maintain the trust from user perspective.
   (b) Off-Chain data management: Manage the data outside of the blockchain provide privacy, is more cheap and scalable but the trust from the user perspective is impacted. There are some technologies that are decentralized as well and are cheapest like IPFS, swarm and other blockchains.
2. Upgradability: As shown in the figure 4 Either by the change of business rules, because bugs or security concerns are essential for all systems to have the capacity to be updated, this is a critical characteristic of the software life cycle.This category helps to save gas.
3. Process/Transactions/Operations:As shown in the figure 3 in this category are listed the patterns that are related with the computations like process, transactions, and any operation. This category is divided in two more categories:
   (a) On-chain Process: Process all the computations inside the blockchain is a very expensive approach and is really hard to scale because the blockchain limitations.
   (b) Off-chain Process: Process all computations outside of the blockchain is very cheap, scalable and provide privacy.
4. Emergency Stop - In centralized systems, when some critical functionality is working bad, and the company could lose money, information, or any other property, the system can be turned off, and all the transactions/processes stop at these moments. In decentralized applications, if a contract is running on the blockchain it can not be stopped by the nature of the system.

The table 1 lists the proposed smart contract patterns by the authors and the characteristics that they accomplish according to the characteristics listed above

All the patterns described by the authors that can be added in the categories listed above are listed in the table 2.

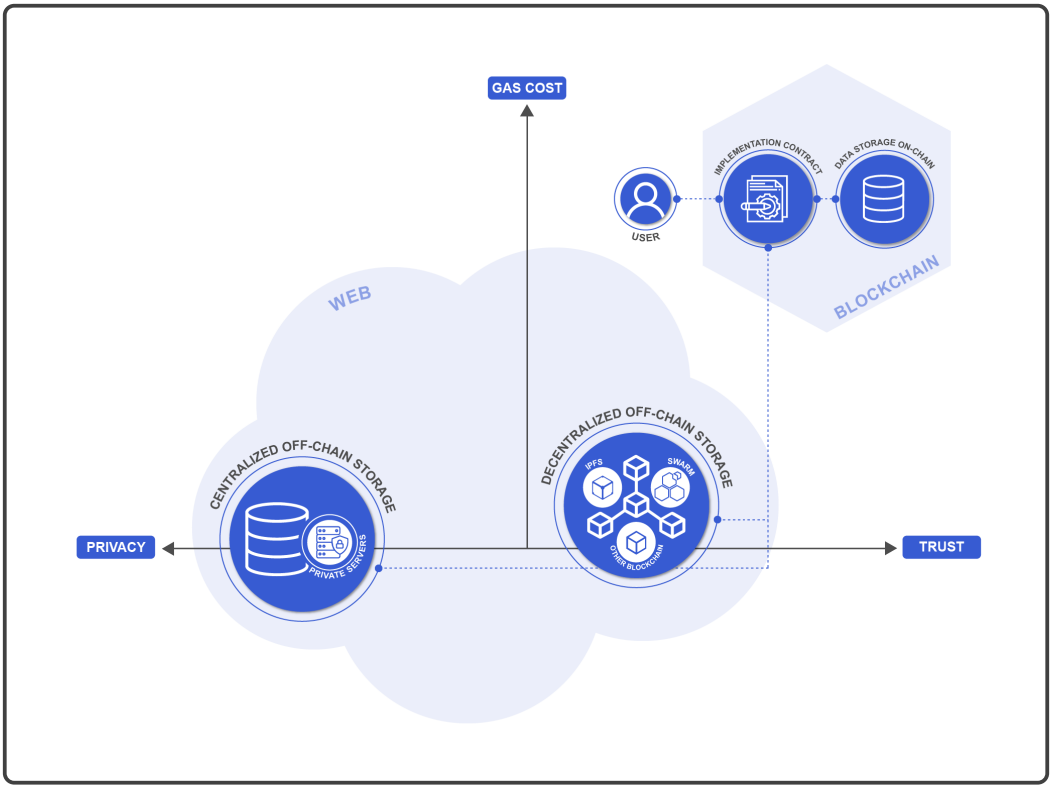


Figure 2: Data storage patterns: Trust/privacy and gas price correlation in

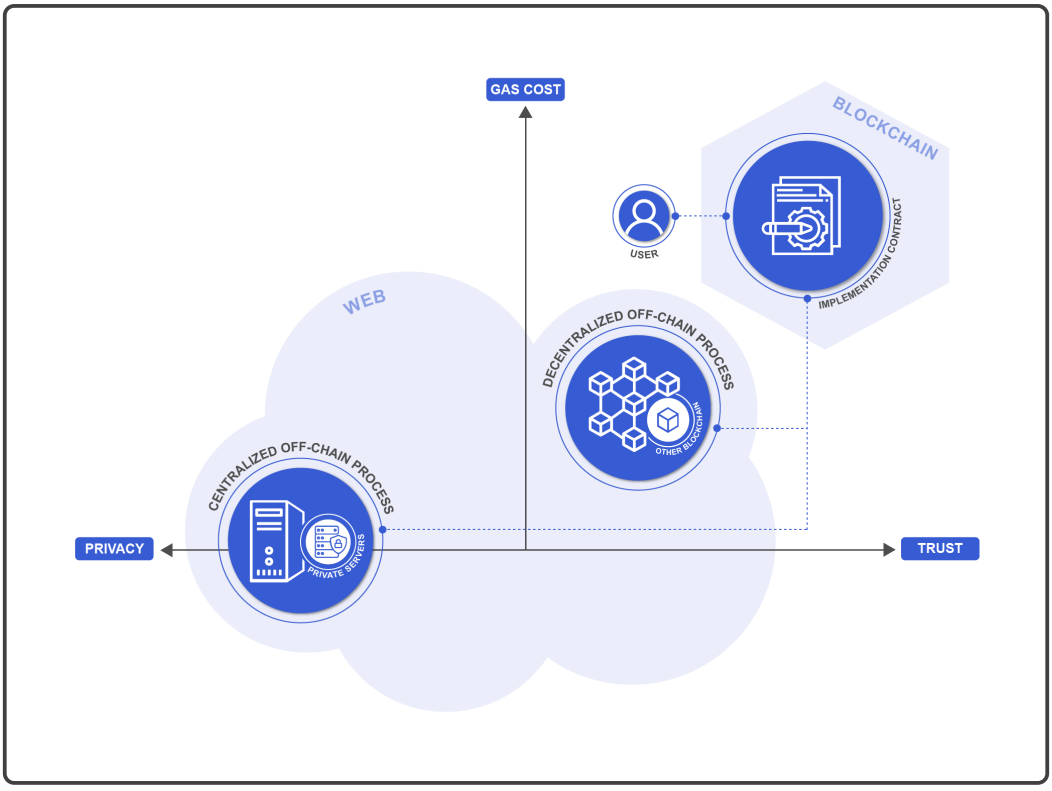


Figure 3: Process/transaction patterns: Trust/privacy and gas price correlation in

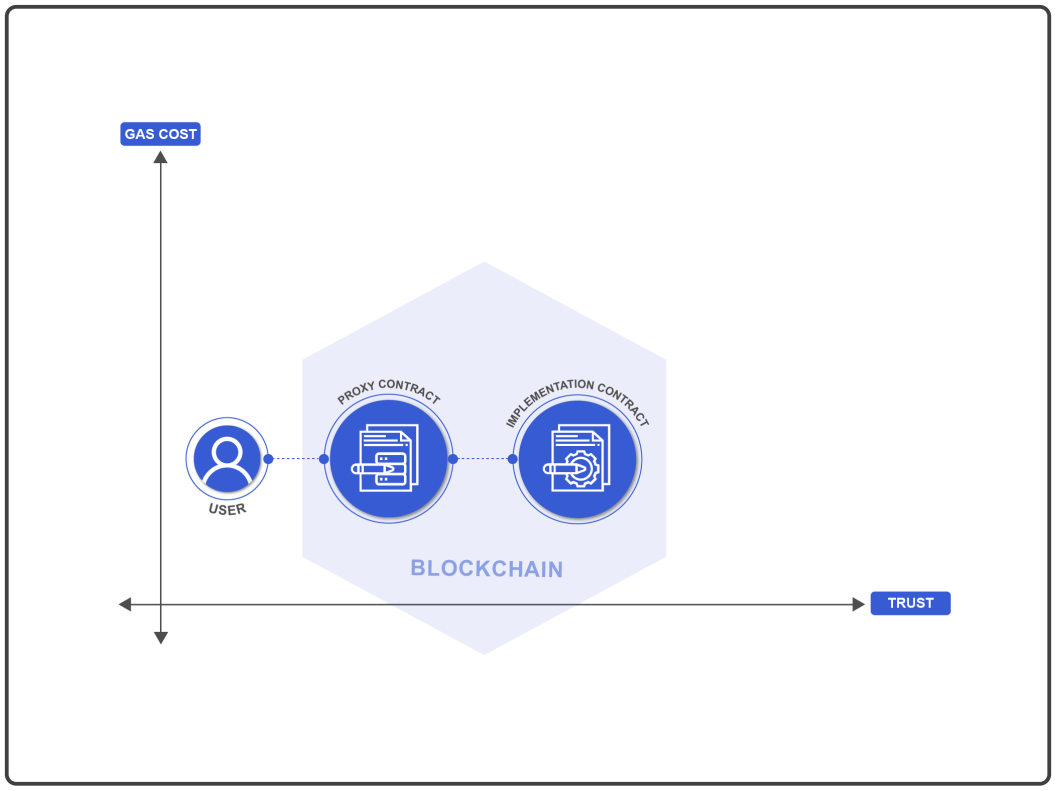


Figure 4: Upgradable patterns: Trust/privacy and gas price correlation

Table 1: Characteristics of actual proposals

Proposed Design Patterns	Upgradability	Process and Data Management In-chain	Process and Data Management Off-chain	Emergency Stop	Considering Gas Consumption
[6]		✓			
[7]		✓	✓		
[9]	✓	✓			
[10]			✓		✓
[8]	✓	✓	✓		✓
This proposal	✓	✓	✓	✓	✓

Table 2: Design Patterns selected

Pattern	Functionality	Category	On-chain/Off-chain	Proposal
Contract Observer	Notify contracts updates from specific contract	Upgradability	On-chain	[6]
Oracle	obtain information out of the chain	Process/Data Management	Off-chain	[7]
Randomness	Deal with the deterministic characteristic of the blockchain providing randomness	Process Management	On-chain	[7]
Termination	Contracts are immutable and remove or delete is not an option, this is a way to disable it	Emergency Stop	On-chain	[7]
Proxy	When a contract is updated a proxy provide a way to maintain the same end-point for the client and just update a variable with the new address	Upgadability	On-chain	[9]
Data Segregation	When a contract is updated all data has to be moved to the new contract, this pattern provides a way to separate the data and only consume the information from this contract	Data Management	On-chain	[9]
Challenge Response Pattern	Verify the final state is expensive on-chain, final state can be verified off-chain	Process Management	Off-chain	[10]
Content-Addressable Storage Pattern	Storage abounding data on chain is expensive	Data Management	Off-chain	[10]
Single Line Swap	Each variable and changes in its content cost gas, this pattern help to have less variables	Process Management	On-Chain	[8]
Use Libraries	put all the code in one smart contract could be expensive and hard for maintenance	Process Management	On-chain	[8]
Delegated Computation Pattern	Execute large amount of operations on-chain is expensive	Process Management	Off-chain	[10]
Packaging Variables	suggest the use of all 256 bits of the minimum variable memory size.	Data Management	On-chain	[8]

4 Design Specification

The focus of this research is not only to suggest design patterns but also to suggest a framework where the developers can select the patterns that fit with the technical requirements.

The framework proposed to take decisions is described in the figure 5 that describes what patterns could be used according with the necessities of the system to be designed. All Dapps must be upgradeable in order to preserve the software life cycle. The first stage is to determine if the operations must be trusted. As the blockchain is a distributed ledger in which nodes confirm public transactions, users could be assured of the authenticity of the transactions. If the process includes essential procedures that an attacker could exploit to drain all tokens, an emergency stop must be applied. The selection of a pattern for trusted operations does not necessitate its usage for all procedures. Certain operations may be performed off-chain, such as on another blockchain with lower transaction costs or by a centralized entity. The user must then determine whether storage is necessary. Consequently, if the data must be kept confidential, the best choice is to keep it outside of a blockchain, because there is no way to manipulate it without disclosing it. Then some data may be stored on a blockchain that could be expensive but provide trust from the user's perspective.

All the patterns listed in the table 2 accomplish with aims of this framework but make a detailed description of each one could be very long and the implementation and implications are very similar by category. One pattern for each category in the flowchart is going to be described as follows:

1. Describe the problem
2. Describe the implications of its implementation
3. Deploy contract with the pattern
4. Run the contract
5. Get the gas consumption(if is applicable)
6. Describe how it works

5 Implementation

5.1 Blockchain

A lot of options in terms of tools for blockchains are available at this time, like networks, code languages, platforms, wallets, and many other things. The most popular are selected to implement the patterns.

The Ethereum blockchain is the most popular platform for deploying smart contracts, is fully documented on Ethereum.org, and is the platform most repeatedly used by the authors described in this study. Due to this, this platform was chosen to install the contracts and measure the Gas efficiency.

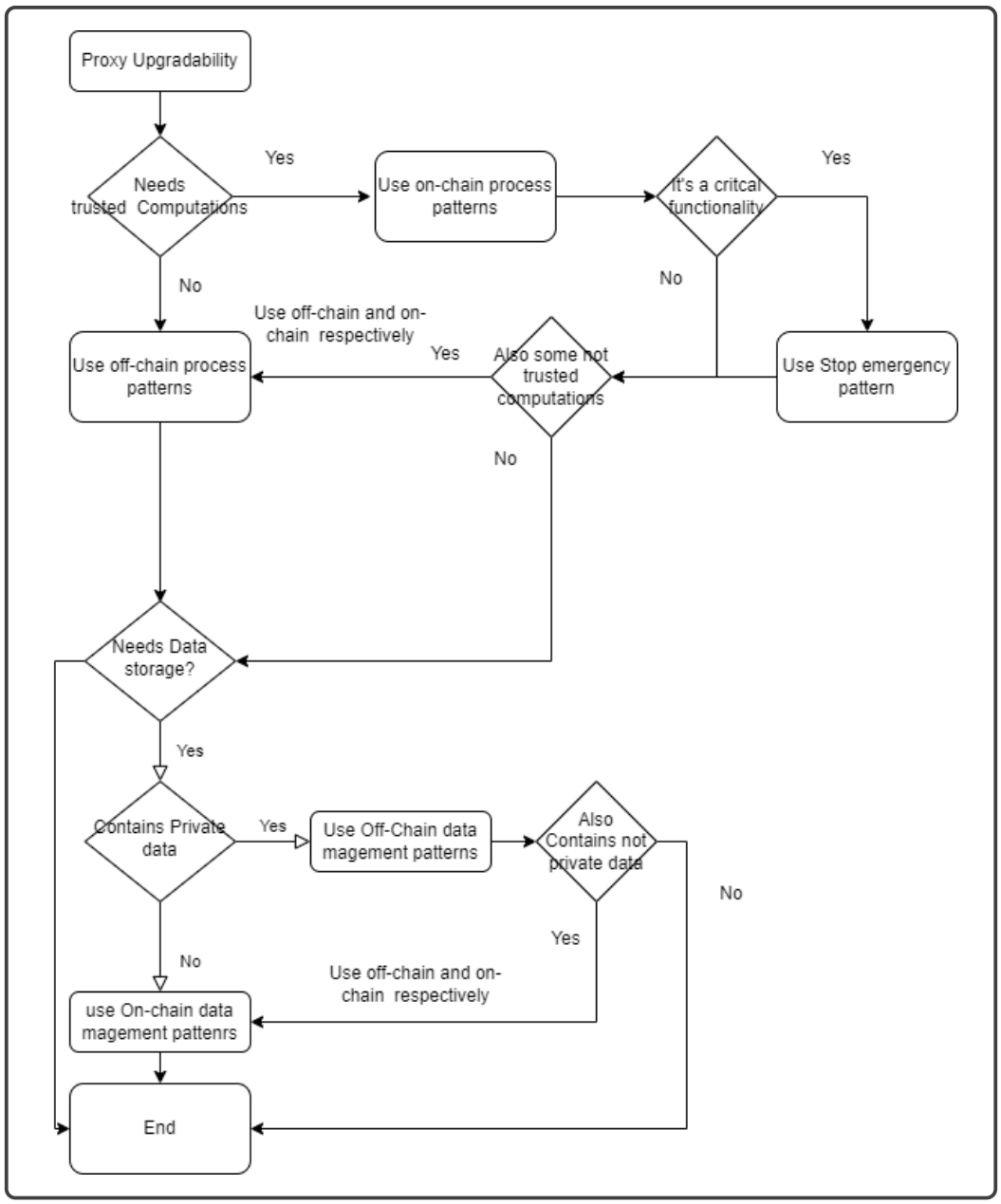


Figure 5: Flowchart to select the pattern needed

Table 3: Proxy pattern implementation fees

Action	Transaction Fees
Deploy Proxy Contract	0.000085057500238161 ETH
Deploy Contract Logic	0.000734677502057097 ETH
Add Address	0.000074772500209363 ETH

5.2 Ethereum network

In the Ethereum ecosystem, several networks exist for multiple uses. Rinkeby, Ropsten, and Kovan are utilized for testing. In overall cases, the gas limit and gas spent by the operation are identical. However, on these networks, no real money is required to make the calculations. Kovan has been chosen as the test network.

5.3 Off-Chain Technologies

Centralized Technologies: If the developer needs to maintain the data private or does not need trusted computations the user can use any private server.

5.4 Wallets

To interact with any blockchain that manages any type of token we need a wallet to help us to manage the account and to perform some operations that we have to execute on the blockchains. By far the most popular is Metamask in all blockchains including Ethereum [4].

5.5 Code Editor

The code editor selected is Remix - https://remix.ethereum.org/, it was created to interact with the Ethereum blockchain and provide multiple functionalities that help with the deployment and execution of smart contracts.

6 Evaluation

6.1 Proxy Delegate

The purpose of this pattern is to add the possibility to update smart contracts without breaking any dependencies, creating an easy way to deal with errors, bugs, or changes in the business rules.

Implications: Its implementation scales the complexity. If it does not have a proper implementation, the chances of adding bugs or unexpected behavior increase as well. From the social point of view, confidence decreases a lot. When the application runs without a proxy contract, the user can know what code is always running. In the case of a proxy contract, the address of the contract is virtually updated, and it can be updated without notifying the user as shown in the figure 6.

The minimum fees needed to implement this pattern are listed in the table 4. The proxy contract only needs to be deployed once. Then all contracts can make requests to this contract. Then the contract that needs to have the ability to be updated is

Figure 6: Proxy Contract

Table 4: Randomness pattern implementation Fees

Action	Transaction Fees
Deploy Randomness Contract	0.00086599 ETH

deployed. This can change all the required times, and the last step in the process is that each time a new contract logic is deployed, the address of this contract needs to be added to the proxy contract. After the implementation of the proxy contract, the only fees that the user needs to handle regarding the architecture of the application (not related to the execution or transactions) are the deployment of new contracts and the fee for updating the address, which could be around 0.000074772500209363. Without this implementation, all the contracts that are making requests to the contract logic should be updated as well, increasing the cost of the whole operation.

6.2 Randomness Pattern

The purpose of this pattern is to provide a way to generate a random number in a deterministic system.

Implications:One option to generate a random number is the use of the oracle pattern to provide it from outside of the blockchain, the oracle option could remove confidence from the user perspective. The second option is to generate it inside the blockchain the problem is that the results are pseudo random numbers that are deterministic and can be repeated after a particular succession.

The only fee needed for this contract is the deployment. This contract could be used only to produce random numbers, and as the contract is not making any modifications or transactions on the blockchain, it does not have any cost to run it. The process of obtaining a random number could be implemented in any other contract. The recommendation is to have this process separated to maintain the upgradability and it could be used for any other contract. The code used in this contract works as follows: First, we take the hash of the last block before the contract transaction, the block timestamp, and one number provided by us. These tree parameters are packed and hashed using keccak256 and using %100 to get the last two numbers. This method provides a random number between 0 and 99; If the developer wants to add more complexity, he can introduce more numbers.

Table 5: Oracle pattern implementation fees

Action	Transaction Fees
Deploy Contract	0.00327778 ETH
Oracle Fee	0.1 LINK Token
Contract Execution	0.00035414 ETH

6.3 Oracle Pattern

The purpose of this pattern is to make a bridge between smart-contracts and the world outside of the blockchain.

Implications: As described above, smart contracts can not directly access off-chain data. To solve this limitation, an oracle is needed, but the first implication is that using single-source data from outside of the blockchain invalidates the decentralization of the smart contract. This problem can be handled using a decentralized oracle that gets data from multiple data sources, and even if one source of data is hacked or has other issues, the contract can still work. It is essential to pay attention that the oracle is secure as the data source is. If the information provided by the data source is corrupted, all the operations related inside the blockchain will also be corrupted. From the social point of view, confidence decreases a lot. Running all the computations inside the blockchain, the user knows what is happening; getting data outside of the blockchain the user does not know what is happening during the data process, and the data could change without notifying the user.

The minimum fees needed to implement this pattern are listed in the table 5. If the contract is used only to return a value from outside of the blockchain, it only needs to be deployed once. If the contract contains some logic that manipulates the data, the recommendation is to split the logic and implement the proxy pattern to maintain the upgradability. The first one is that the developer creates their own oracle, but the infrastructure needed to keep the decentralization and remove one point of failure is expensive. The second option is to use an oracle service, which is usually decentralized, but depending on the oracle, it can require extra fees. In this case, the contract uses the Chainlink oracle. To use the Chainlink oracle, we need to send a LINKS token to the contract, which will be used at the moment of execution. So, when the contract is executed, there will be Ethers used for the execution inside the blockchain and Link tokens for the chainlink oracle.

Even when using an Oracle service requires extra fees, it could be cheaper than performing extensive computations inside the blockchain.

6.4 Emergency Stop

The purpose of this pattern is to add the option of deactivating a critical smart-contract functionality in case of emergency.

Implications: The only concern about these patterns is from the social point of view; the users need to trust in the entity that has the control to stop the contract, but there is always the possibility that the stop is executed in a malign way.

The fees to implement this pattern are described in the table 6 and they do not include costs to deploy it because this pattern is added to the critical functionality of another contract. The fees to add these functions to the same contract are the minimum.

Table 6: Emergency stop pattern implementation fees

Action	Transaction Fees
Stop Contract	0.0000662 ETH
Resume Contract	0.00006625 ETH

Table 7: Eternal storage pattern implementation fees

Action	Transaction Fees
Deploy Contract	0.00059521 ETH

Then the fees paid to stop the functionality are 0.0000662, which is very low compared to the losses resulting from a malfunction. Then the contract can be reactivated. Two more functions can be added to this pattern: emergency withdrawal and deposit. Both functions can be executed in case of an emergency when the contract is stopped.

6.5 Eternal storage pattern

The purpose of this pattern is to add the possibility of keeping the storage after the upgrade of a smart contract.

Implications: The main advantage of this pattern is the removal of data migration after a contract update. Implementing this pattern can add more complexity to the decentralized application because it needs decentralized calls, and the decentralized calls must be managed with care because they can trigger unexpected behavior. From the social point of view, trust from the user perspective decreases because the store owner can implement a new version of the contract to manipulate the data to his advantage and redeploy another version without the user advice.

The only fee needed for this pattern as described in the table 7 is the contract deployment, all other fees like the price for storing data, depend on the amount of data that needs to be stored and once the data is stored, the logic contract can be updated many times as needed, and all contracts can still use the same storage.

6.6 Variable Packing

The purpose of this pattern is the optimization of gas consumption at managing variables.

Implications: This pattern can be implemented inside the contract that was designed with any other pattern, and the main benefit is the amount of gas that can be saved during the lifetime of the contract. The first concern is that if this pattern is not correctly implemented, the amount of gas used could be huge. This pattern only works with statically sized storage variables and inside structs, not with function parameters or dynamic arrays. As we can see in the table 8 is more costly for the Ethereum virtual machine to perform the contract deployment with this pattern. From the social point of view, the moment of ordering the variables could reduce the readability because, usually, the variables are declared in a logical order, and changing that order could make it hard to audit the code.

As shown in the table 8 the contract deployment with the pattern is more expensive, but the extra fees paid can be recovered during the lifetime of the contract. Basically,

Table 8: Variable Packing pattern implementation fees

Action	Packed Variables	Without Packing
Deploy Contract	0.00052149ETH	0.00042607ETH
Storage	0.00011486ETH	0.00049653ETH

this pattern is focused on use more efficiently the storage slots created by the EVM which is 32bytes (256bit). Each time when a uint is declared the EVM will be stored in these slots. If we declare:

1. uint128 a;
2. uint256 b;
3. uint128 c;

The EVM will use one slot for the first one, then the second one does not fit in the first slot, so another slot is used, and then for the last variable, as the second slot is full, a new slot will be used. In total, we used three slots. But if the variables are declared as follows:

1. uint128 a;
2. uint128 c;
3. uint256 b;

Then the first two variables can fit into one slot and the second one into a second slot.

6.7 Discussion

In general, the implications of the use process or storage inside or outside the blockchain are very similar. If the process/storage are inside the blockchain the transaction could implicate more fees to pay for any modification, but the user could trust more in the execution of the contracts with the exception of the proxy contract. In the case of implementing off-chain patterns the main concerns are the trust from the user point of view but the transaction would be cheap, fast and scalable.

7 Conclusion and Future Work

May the patterns described by the authors be employed as part of a distributed backend application? , Yes. All the proposed patterns deal with different disadvantages of the blockchain, even when the pattern does not directly relate to the characteristics that we are trying to accomplish from the most common centralized applications. The flowchart proposed could help the developers make the best architectural decisions and with the inputs of the detailed analysis like gas consumption and trust from the user perspective they can decide what is more important, trust or gas consumption. With this information, they can create better systems, and provide a better user experience.

In the future work could be analyzed the rest of the patterns in the table 2 to provide more inputs to the developers and add other patterns that do not necessarily meet the

characteristics of the centralized application but help with the weaknesses of the blockchain extending the framework and the flowchart in the figure 5.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008, no JCR or CoreRank found, Accessed on 06.04.2022.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014, no JCR or CoreRank found, Accessed Online on 06.04.2022.

[3] H. Govind and H. González-Vélez, "Benchmarking serverless workloads on kubernetes," in *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021*. Melbourne, Australia: IEEE, May 2021, pp. 704–712, CORE2021 Rank: A.

[4] F. Blum, B. Severin, M. Hettmer, P. Hückinghaus, and V. Gruhn, "Building hybrid dapps using blockchain tactics -the meta-transaction example," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–5, jCR Impact factor 2021: 3.28.

[5] M. Danelutto, G. Mencagli, M. Torquati, H. González-Vélez, and P. Kilpatrick, "Algorithmic skeletons and parallel design patterns in mainstream parallel programming," *International Journal of Parallel Programming*, vol. 49, no. 2, pp. 177–198, 2021, jCR Impact Factor 2020: 1.3.

[6] Y. Liu, Q. Lu, X. Xu, L. Zhu, and H. Yao, "Applying design patterns in smart contracts," in *Blockchain – ICBC 2018*, S. Chen, H. Wang, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2018, pp. 92–106, jCR Impact factor 2021: 3.28.

[7] B. Massimo, P. Livio, R. Kurt, B. Joseph, M. Andrew, P. Y. Ryan, T. Vanessa, B. Andrea, S. Massimiliano, P. Federico, and J. Markus, *An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns*. Cham: Springer International Publishing, 2017, CORE2021 Rank:A.

[8] L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, and D. Tigano, "Design patterns for gas optimization in ethereum," in *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, London, 2020, pp. 9–15, CORE2021 Rank: C.

[9] V. C. Bui, S. Wen, J. Yu, X. Xia, M. S. Haghighi, and Y. Xiang, "Evaluating upgradable smart contract," in *2021 IEEE International Conference on Blockchain (Blockchain)*. Melbourne: IEEE, 2021, pp. 252–256, jCR Impact factor 2021: 3.28.

[10] J. Eberhardt and S. Tai, "On or off the blockchain? insights on off-chaining computation and data," in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds. Cham: Springer International Publishing, 2017, pp. 3–15, no JCR or CoreRank found, Accessed Online on 06.04.2022.

[11] A. M. Antonopoulos and G. Wood, *Mastering ethereum: building smart contracts and dapps*, 1st ed. Sebastopol: O'reilly Media, 2018. [Online]. Available: https://github.com/ethereumbook/ethereumbook

[12] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," in *Conference on the Theory and Application of Cryptography*. Berlin, Heidelberg: Springer, 1990, pp. 437–455, CORE2021 Rank: A. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-38424-3_32

[13] M. P. Rodríguez Bolívar and H. J. Scholl, "Mapping potential impact areas of blockchain use in the public sector," *Information Polity*, vol. 24, no. 4, pp. 359–378, 2019, jCR Impact Factor 2021: 1.031.

[14] N. Szabo, "Formalizing and securing relationships on public networks," *First monday*, 1997, no JCR or CoreRank found, Accessed Online on 06.04.2022.

[15] ——, "The idea of smart contracts," *Nick Szabo's papers and concise tutorials*, vol. 6, no. 1, p. 199, 1997, no JCR or CoreRank found, Accessed Online on 06.04.2022.

[16] ——, "Secure property titles with owner authority.(1998)," *Online at http://szabo.best.vwh.net/securetitle.html*, 1998, no JCR or CoreRank found, Accessed Online on 06.04.2022.

[17] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0. 8.13," *Whitepaper*, 2018, no JCR or CoreRank found, Accessed Online on 06.04.2022. [Online]. Available: https://solana.com/solana-whitepaper.pdf

[18] E. project, "smart contracts ethereum.org 2022," ethereum.org/en/developers/docs/smart-contracts, 2022, accessed Online on 06.04.2022.

[19] Solana, "Solana docs 2022," https://docs.solana.com/, 2022, accessed Online on 06.04.2022.

[20] P. Hegedűs, "Towards analyzing the complexity landscape of solidity based ethereum smart contracts," *Technologies*, vol. 7, no. 1, 2019, no JCR or CoreRank found, Accessed Online on 06.04.2022. [Online]. Available: https://www.mdpi.com/2227-7080/7/1/6

[21] M. Wöhrer and U. Zdun, "Devops for ethereum blockchain smart contracts," in *2021 IEEE International Conference on Blockchain (Blockchain)*, 2021, pp. 244–251, jCR Impact factor 2021: 3.28.

[22] V. Buterin *et al.*, "Ethereum white paper," *Ethereum project*, vol. 1, pp. 22–23, 2013, no JCR or CoreRank found, Accessed Online on 06.04.2022.