كلية الهندسة بشبرا
FACULTY OF ENGINEERING AT SHOUBRA
ERJ
PRINT ISSN 3009-6049
ONLINE ISSN 3009-6022

ENGINEERING RESEARCH JOURNAL (ERJ)
Volume (53),Issue (3)
July 2024, pp:10-21
https://erjsh.journals.ekb.eg

Towards a Secure Blockchain-based E-Government Framework in Egypt: A Case Study

Manal Mansour [1,*], May A. Salama [1], Hala H. Zayed [2], Mona F.M. Mursi [1]

[1] Faculty of Engineering, Egypt University of Informatics, Cairo, Egypt.
[2] Faculty of Computers and Artificial Intelligence, Benha University, Egypt.
**Corresponding author*

E-mail address: manal.abdelmawgood@feng.bu.edu.eg , may.mohamed@feng.bu.edu.eg , hala.zayed@feng.bu.edu.eg , mona.mursi@feng.bu.edu.eg

Abstract: Blockchain stands out as one of the most promising technologies due to its potential to enhance security, transparency, and privacy. Consequently, blockchain-based applications serve as a trust-building mechanism among involved stakeholders. Several governments have recently recognized these advantages and have initiated the integration of blockchain into their public service sectors. Notably, Egypt has yet to explore the potential of blockchain within its e-government services. This research introduces a novel framework, along with its empirical implementation that aimed at establishing a scalable model for two e-government services in Egypt's land registry and e-will management. The framework's primary objective is to create a system that is transparent, immutable, and secure. Furthermore, solutions are presented to address the technical challenges, security and privacy issues encountered during the implementation phase. This marks the initial proposal of a blockchain-based public service framework for the Egyptian government.

Keywords: E-Government, Blockchain, Smart Contract, Real Estate, E-Will.

1. Introduction

In recent years, many governments around the world have recognized the potential benefits of digital technologies and have sought to leverage these technologies to improve the delivery of public services, from online portals and mobile applications to the use of blockchain and other emerging technologies. Governments are increasingly exploring new ways to provide citizens with faster, more convenient, and more secure access to essential services. In this context, a framework is proposed, emphasizing two crucial Government-to-Citizen (G2C) services [1], namely real estate and wills. The framework can be extended to include a wide range of additional services. Its flexibility is one of its key strengths, allowing for adaptation to meet the needs of different contexts and service areas. Real estate and wills were selected as the primary services in the proposed framework for several reasons.

Upon examining the current state of these services in Egypt, several challenges were uncovered. These include issues such as counterfeit property contracts, disputes over property ownership, and individuals manipulating sale dates to circumvent government regulations. Moreover, problems related to property resale and an increase in fraudulent activities have been observed. The government lacks a comprehensive record of property transactions and sales history and faces difficulties in collecting taxes from property owners due to the diverse range of contract types and selling methods. Adding to these challenges, property owners may pass away without leaving information for their heirs about the location of vital documents related to their property.

Another significant concern that emerged in 2020 is the challenge of wallet credentials. This problem arose in 2020 when Chainalysis, a crypto research company estimated that roughly a fifth of existing bitcoins [2] at that time, with a value of more than $175 billion, had been lost due to the death of the owners. In blockchain, there is no way to access their wallets except with the private key. Effectively utilizing E-will in blockchain has the potential to deliver significant benefits in resolving these issues.

In this paper, A novel method is proposed for connecting real estate and wills by harnessing the capabilities of blockchain technology and smart contracts, while leveraging the unique features of blockchain, such as its decentralized, transparency, and immutable nature. The proposed framework offers a transparent, coherent, and secure method for transferring ownership of real estate, whether through inheritance or by the selling process during the owner's life. The proposed framework has the potential to bring about a revolutionary change in the way real estate transactions are conducted. This paper is organized into six distinct sections, each serving a specific purpose in our comprehensive examination of the subject. Section 2 provides background information and context regarding the main topics under consideration. The proposed framework, including its key features, advantages, and potential limitations, is introduced in Section 3. In Section 4, a dedicated segment is presented, focusing on the tools utilized and the available alternatives. Section 5 of this work delves into the security analysis and presents the proposed solution. In this section, a comprehensive assessment of the security measures is conducted, followed by the proposed solution that aims to address any identified vulnerabilities and enhance overall framework security. Finally, section 6, summarizes the main findings and contributions of the research, discusses any limitations encountered, and provides recommendations for future studies or practical implementations.

2. BACKGROUND

Blockchain technology is a decentralized and distributed digital ledger that is used to record transactions across many computers in a secure, transparent, and immutable way. It consists of a chain of blocks that contain data. Each block in the chain is linked to the preceding block, and once a block is added to the chain, it cannot be altered or deleted. Bitcoin is the most popular and known platform called the first generated platform [2] and [3] primarily used for financial deals only with 1 Megabyte (MB) blocks. The blocks are linked together through a complex cryptographic verification process, to form the immutable chain.

2.1 Smart Contracts

Smart contracts are considered the second-generation evolution of blockchain technology, offering a wide range of applications extending beyond financial transactions. These self-executing computer programs automatically enforce the terms of agreements between parties [4]. Many DApps use smart contracts such as supply chains, voting systems, healthcare, and real estate beyond others [5] and [6]. Numerous platforms support smart contracts such as Ethereum, which offers a degree of standardization, Hyperledger, and Corda among other platforms [19].

2.2 Potential Risks and Threats Related to Smart Contracts

Many reported weaknesses can potentially compromise the security and functionality of smart contracts [7] and [8]. Table 1 summarizes the smart contracts' weakness classifications (SWC) allowing developers to take proactive measures to mitigate such risks and safeguard their projects.

3. PROPOSED FRAMEWORK

Before starting this phase, a meeting with the real estate professionals who specialize in providing real estate services in Egypt was held to gather information and conduct a thorough analysis of the data at hand. That resulted in a comprehensive understanding of the real estate industry in Egypt and its various intricacies.

3.1 Analyzing Challenges in the Real Estate System and Implications of Writing Wills

The traditional property registry system contains several conceptual stages, such as housing evaluation, document compilation, main contract execution, money transfer, and registration [12] and [13]. All these steps make the complete procedure complicated and costly. Additionally, a lack of synchronized information due to inadequate departmental cooperation results in inconsistency and incompatibility. On another hand, there are several challenges to writing and executing wills so, the testator has to consult with legal professionals who specialize in wills and inheritance matters. They guide navigating the legal framework, ensuring the validity of wills, and addressing any potential issues that may arise during the distribution process. The following points summarize these challenges [14]. 1- Lack of Regulation: There is no specific regulation for writing wills, so it is important to seek legal advice from professionals who are knowledgeable about the local legal system. 2- Complex Legal Framework: The legal framework surrounding wills and inheritance in Egypt can be complex and may require a thorough understanding of Egyptian laws. In addition to these problems, regarding digital cryptocurrency and digital assets, if the location and private key of the cryptocurrency wallet are not known, the funds may be lost forever after the owner's death [15].

3.2 Building Framework

The proposed framework is designed using Visual Studio code, Node js v16.17.1, Solidity ^0.8.9 [16], and Hardhat [17][18], which is available as a npm package and comes with many tools that facilitate development along with a test environment that allows the developers to test the contracts locally. The primary objective of the proposed system is to build a framework for e-government services by establishing a connection between two distinct services, namely real estate and wills. Each service can be applied separately or connected through the utilization of blockchain and smart contract technology. The system has been designed to encompass a single pivotal contract; the government contract, which is deployed solely once, in addition to two supplementary model contracts; the real estate contract, and the e-will contract. Notably, the latter two contracts are subjected to specific addresses that are meticulously set by the government.

3.3 Smart Contract Emulation of Government Processes

The government contract is deployed on the Ethereum network during setup, using predefined addresses for key roles such as government, real estate, and judicial managers. These trusted nodes possess the authority to deploy, revise, update, approve, and reject citizen requests within the contract. Table 2 shows all members and variables of the government contract.

3.3.1 Contract Actors, Functions, and Events

'REPublicityManager' is the address of the manager(s) who is managing the real estate functionality; this node is trusted. The second variable in table 2 is the 'IdtoRealEstatemap'. This variable contains a map between each ID and a corresponding array of properties mapping (string => address[]) public 'IdtoRealEstatemap'. Using a map, this step solves significant problems in serving the requests of citizens. In Solidity, maps use a different approach to store data, which saves time and reduces gas consumption compared to using an array for handling large amounts of data. Maps in solidity use the hash table with the look-up process method. It works differently here. Provide the map with a key, citizen ID, and then the key is introduced to a hashing function and produces the corresponding value, see Figure 1.

Table 1 Smart Contract Weakness Classifications [9]

Vulnerabilities	Code	Description
Function Default Visibility	SWC-100,108	Public default declaration.
Integer Overflow and Underflow	SWC-101	Exceeds the size of data types.
Outdated Compiler Version	SWC-102,111	The outdated compiler may pose issues.
Unchecked Call Return Value	SWC-104	Unexpected behavior in the program.
Unprotected Ether Withdrawal	SWC-105	Malicious withdraw some Ethers.
Unprotected SELF-DESTRUCT	SWC-106	Insufficient access controls.
Reentrancy [10]	SWC-107	Take over the control flow.
Uninitialized Storage Pointer	SWC-109	Point to unexpected storage locations.
Transaction Order Dependence [11]	SWC-114	Race condition vulnerability.
Shadowing State Variables	SWC-119	Same var name with inheritance
Insufficient Gas Griefing	SWC-126	Revert from any sub-calls
Dos with Block Gas Limit [9]	SWC-128	The cost of executing a function exceeds the block gas limit.
Presence of unused variables	SWC-131,132	Gas consumption
.Message call with the hard-coded gas amount	SWC-134	Use a fixed amount of gas
Dead code	SWC-135	Code With No Effects

Table 2. Government Contract

Government Contract		
Variables		
REPublicityManager	Address	Address of the manager/s who are managing the real estate
JudicialAuthorityManager	Address	Address of the manager/s who are managing and approving wills
IDtoRealEstate	Map	Map between each citizen and the array of his real estates
IDtoEWill	Map	Map between each testator and his will
Functions		
Government	The constructor function sets the manager address on deploying government contract	
Create RealEstate	Factory function which deploys new real estate contract	
ApproveRE	Approve citizen request	
RejectRE	Reject citizen request - Destroy the contract	
InformOwnershipTransfere	Update the ledger with the new owner after the sell/inherit process	
Create EWill	Factory function deploys a new EWill contract	
ApproveEWill	Approve Creating EWill	
RejectEWill	Reject Creating EWill	
Execute EWill	Call MarkAsDead in EWill contract	

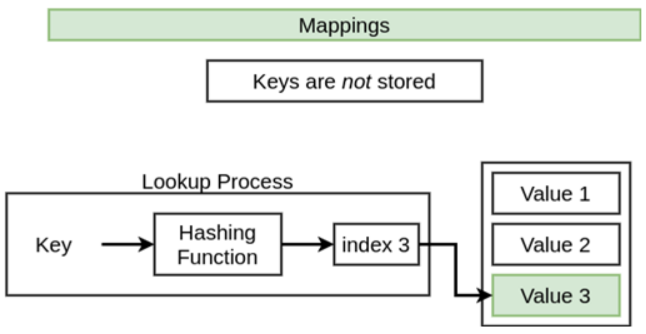


FIGURE 1: Maps in solidity

This method reduces the time consumed in the search process as the arrays use a linear search methodology with O(n), which is supposed to be more than a million entries in this contract. The search process consumes many gases, which consequently raises the cost of dealing with the contract. Otherwise, the map uses a constant time search, which implies that the amount of data doesn't matter. One million citizens take the same time as a single entry to find their corresponding data O(1). After deploying a government contract, the citizens can send requests to this contract. Each request is first mined and peer-reviewed by the real estate publicity managers.

CreateRealEstate: In the 'CreateRealEstate' step, a citizen requests property registration and ownership proof from the government by submitting his ID and property details. The government promptly deploys a real estate contract with a pending state, shares the contract address with the citizen for status checking, and updates its 'IdtoRealEstatemap' by adding the contract address to the ledger. If the government rejects the state, the 'reject' function is invoked, leading to the destruction of the real estate contract. Approve: This function receives the generated contract address and allows the real estate publicity manager to revise, approve the citizen request, and confirm the addition to the 'IdtoRealEstatemap'. Reject: This function receives the generated contract address, rejects the citizen request, destroys the contract, and removes the temporarily created entry from the 'IdtoRealEstatemap'. Figure 2 shows the process of creating real estate. Algorithm 1 illustrate the steps.

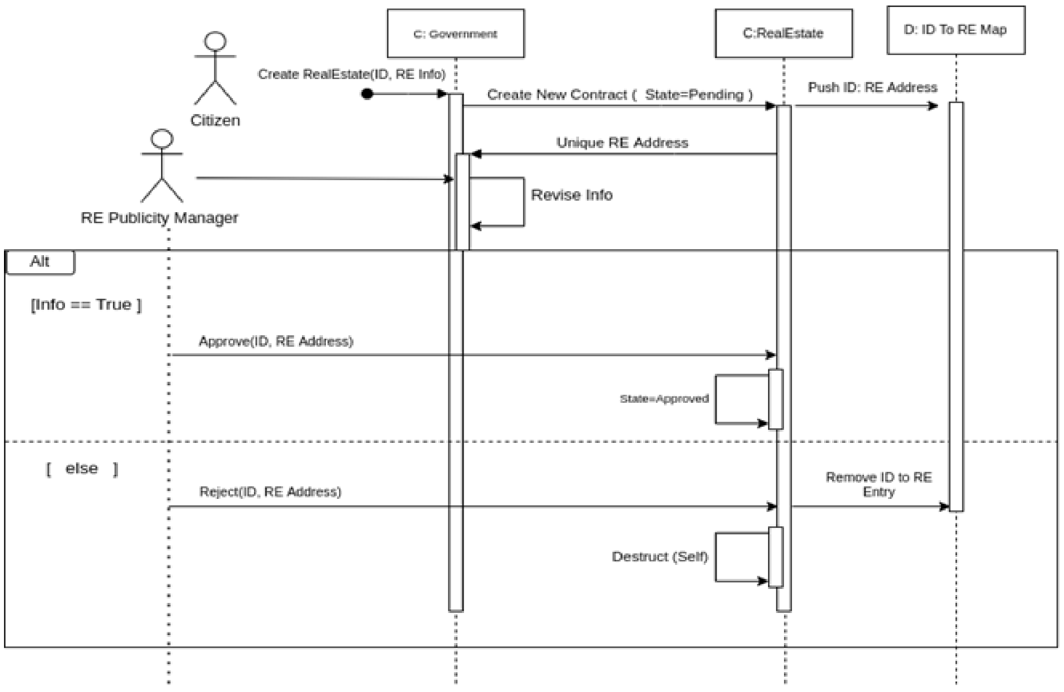


FIGURE 2: Create Real Estate

Algorithm 1: Create Real estate Contract

Input:
Government Contract G← Citizen(x) (ID, Real estate RE Information)
Process:
address newRealEstate = address (new RealEstate(id, info, Citizen(x))); //Government-Restricted Deploy RE
idToRealEstateMap[id]. push(newRealEstate); //Update the Map
Output:
Citizen(x) ← return newRealEstate address with state Pending;
if (request valid)
State← Accepted.
else
State ←Rejected + Self Destruct

3.4 Representing Real Estate as a Smart Contract

This contract is a model contract. The government deploys a copy from this contract for each real estate proof request. Table 3 Summarizes the real estate contract variables and functions.

3.4.1 Contract Actors, Functions, and Events

Government Contract Address: The address of the government contract. This address only is allowed to deploy real estate contracts and is verified for government-restricted functions. Property Information: All the descriptions and related information for the asset, which are required to be recorded to create its contract for the first time. This information has to be set in the constructor. Transaction: Struct variable contains the information of the seller and buyer, in addition to the ownership transfer method either the buy-sell process or inheritance. State: This variable contains the status bending of this contract before government approval. After the acceptance, the contract status is changed to accepted. Otherwise, the contract is destroyed if the request is rejected. Table 4 contains all the members of the real estate struct.

Table 3. Real Estate Contract

Real Estate Contract		
Variables		
Property Info	Struct	Contains property information
Government contract Address	Address	Allow government-restricted function
Owner Address	Address	Address for the property owner
Owner I D	String	Owner ID
Transaction	Struct	Contain transaction information
Transaction-History	Array	Save the sequence of ownership transfer
state	Struct	Contract state [Pending-Accepted-Rejected]
Functions		
Constructor	Government restricted	
OfferProperty	Owner-restricted	
Unoffer	Cancel the sell offer-owner restricted	
WillingtoBuy	Called by all the citizens who aim to buy the property.	
Buy	payable-atomic function-Winner restricted	

Table 4. Struct members in the RE contract

Struct members in RE contract		
Transaction Struct		
From ID	string	ID for the old owner
From Address	Address	Address of the old owner
To ID	string	ID for the new owner
To Address	string	Address of the new owner
Date	uint256	timestamp of the transaction
Method	enum	sell, inherit
Property Information		
Description	string	
Unit-No	string	
Building No	string	
Street	string	
City	string	
GPS-Location	latitude and longitude	

Transaction history: Array that contains the history of this asset from the first owner to the current owner. Offer-the-property: This function exists in the real estate contract and is owner-restricted which means that no one even the real estate publicity manager can call this function from the contract. The function should accept the value of the

asset to be declared later. UnOffer-the-property: Cancel the offer request. This function is also owner-restricted. Willing-to-Buy: This function is called by citizen(s) willing to buy the offered asset. The winner is determined by the government. The priority is set for the citizen who has the preemption right.

Buy-Property: This is an atomic payable function, called by the winner from 'willingtobuy' function only. The winner sends a predefined amount of money to the function performing these atomic steps to execute all of these steps or undoes them completely. Initially, the function takes the money and transfers it to the owner of the assets, and sets the transaction struct information. The function calls the inform ownership change function from the government contract. This function verifies the existence of the old ownership in the 'IdtoRealEstatemap' If it exists, the asset is transferred to the new owner by adding the real estate's address to the 'IdtoRealEstatemap'. Subsequently, the transaction history of this contract is updated by appending this transaction to the 'TransactionHistory' array, which preserves the sequence of ownership transfers associated with this contract. If any of these steps stop working for any known or unknown reason, the function reverts. Figure 3 summarizes the selling process. Algorithm 2 illustrates the selling steps.

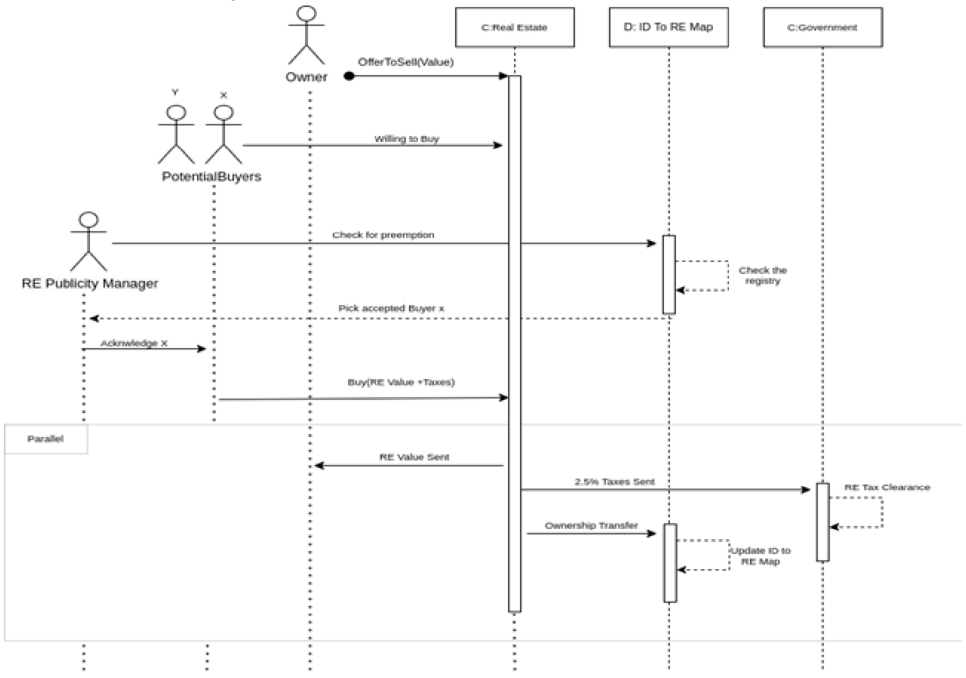


FIGURE 3: Selling Process.

Algorithm 2: Buy / Sell Process

```
Input:  OfferProperty Status ← Open (Owner-restricted)
Process:
Wiliingtobuy ← users // user intended to buy the asset
buyers.push(users);
acceptedBuyer = buyer; // Picked by Government Using Preemption law
payable(owner_address)←transfer(Offer.value);
payable(Government_address)←transfer(Tax.value);
owner_address = acceptedBuyer;
      owner_id = newOwnerId;
      history.push(newTransaction);
   method: TransactionMethod.SELL,
Output:
Transaction history  and IdtoRE map ← Updated
Taxes ← collected.
The transaction method in the RE contract is set to 0 (transfer is done by the selling process).
```

Atomic Done or Revert

3.5 Implementing Citizen Wills as Smart Contracts

The testator sends a request for the government to create his own will,the 'IDtoEwill' is one to one relationship so if the government revises the request and ensures that this ID didn't issue a will request it approves the citizen request and deploys an E-will contract. The testator then be able to add money beneficiaries, add real estate beneficiaries, add priorities for his wishes, guaranteeing that his document remains immune to forgery, and ensure automatic distribution of his belongings according to his wishes eliminating the need for an attorney or considering where to save the will. Table 5 contains all variables used for the e-will contract.

3.5.1 Contract Actors, Functions, and Events

Government contract address: This address is the only one allowed to deploy e-will contracts and is verified for government-restricted functions.

TestatorID and TestatorAddress: These are the identity and the address of the testator. The government inspects and revises it to ascertain the authorization and uniqueness of the e-will creation request. willMoneyEntry[]: Array of beneficiaries to which the testator wants to distribute his money after his death. willRealEstateEntry[]: Array of beneficiaries to which the testator wants to distribute his real estate after his death. Application State: This variable contains the status of the testator's request for deploying his will with state pending. After the judicial authority decision, the status may be changed to approved and proceed to blockchain or rejected then destroy the contract.

The constructor function takes the address of the judicial authority manager to set the restricted functions. Create e-will function: This function is a factory function that deploys a new e-will contract by the judicial authority manager. The function delivers a request from the citizen and responds with an address for the deployed e-will with status pending. The testator should check the address of his will till the new status is delivered, approved, or rejected by the judicial authority manager. Figure 4 shows the creation of the ewill process.

Approve e-will: It approves the creation of the will after revising the existing testator address. Reject e-will: It rejects the testator request for any reason. e.g. The citizen has created another will. So, his ID is related to the old will address. Execute e-will: This function is a judicial authority manager restricted. It takes the testator's ID after his death, checks the existence of the will, and calls the 'MarkAsDead' function from the related e-will contract. 'MarkAsDead' function then returns real estate entries and an array of beneficiaries. The government contract revises the ledger to ensure the ownership of this real estate from 'IDToRealEstatMap' and then transfers its ownership to the beneficiaries. Transferring the ownership by the 'executeEwill' function then changes the struct member to 'inherit'. Figure 5 illustrates the executeEWill function. Algorithm 3 shows the steps for executing the will.

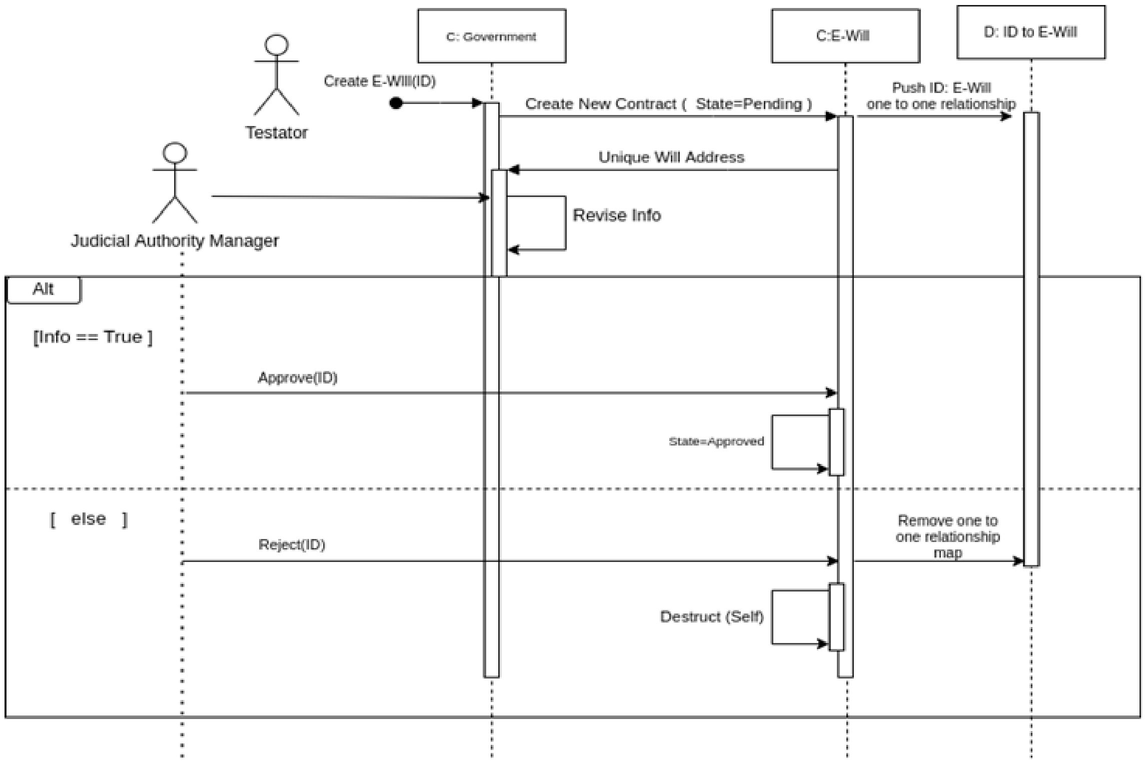


FIGURE 4: Create E-Will

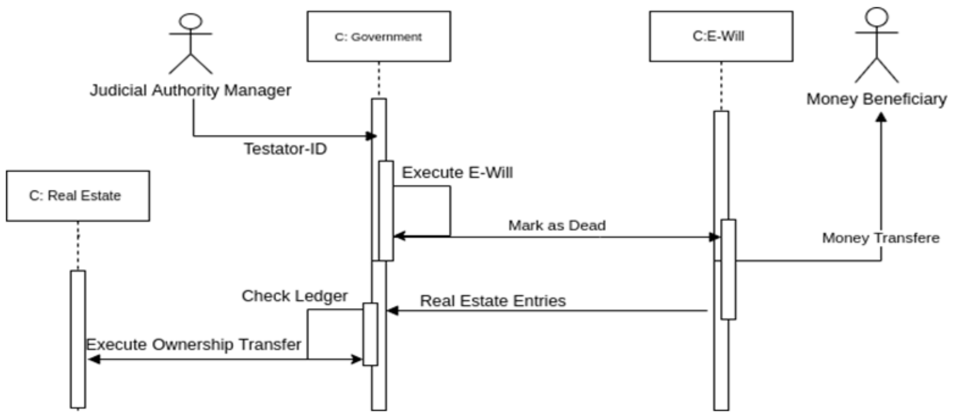


FIGURE 5: Execute E-Will

```
Algorithm 3: Mark As Dead Process
Input:
 MarkAsDead()← Execute E-will() Judicial Authority manager Restricted.
Processes:
1- MoneyTransaction
Sort Will Money Entries By Priority \\ Recorded by the testator
for (; i < willMoneyEntries.length; ++i)
payable(willMoneyEntries[i].to).transfer(willMoneyEntries[i].value);
2- RETransaction
        benefeciaryAddress←REContract Transfere
          method: TransactionMethod.INHERITANCE,
          history.push(newTransaction);
           date: block.timestamp
Output:
1- Money Transferred to beneficiaries.
2- IdtoREMap updated with the new owner.
3- Transaction method in RE contract set to 1.
```

4. TESTING AND ANALYSIS ROAD MAP

Building the framework with Solidity language doesn't guarantee its security, functionality, or efficiency, Code testing should be conducted first before deploying the code on the net.

Where it engages with real users and assets. This section intensively investigates various best practices and tools essential for rigorously testing the Solidity code, ensuring a comprehensive evaluation of its performance, and mitigating potential issues.

4.1 Used Tools, Frameworks, and Technologies

In this section, the steps taken after constructing the framework will be outlined. The tools used for testing and analyzing the framework's performance to ensure its strength and reliability will be addressed. Breaking down the testing process and explaining the role of each tool. Figure 6 summarizes the steps.

2. Code Analysis Tool: To analyze the framework, the initial step selects software tools designed to identify security issues and vulnerabilities associated with smart contracts. Code analysis tools for Solidity include MythX, Solc, and oyente, which can aid in mitigating the identified issues.

3. Choose a testing framework: The initial stage in testing Solidity code involves selecting a testing framework that aligns with specific requirements and preferences. A testing framework serves as a software tool facilitating the creation, execution, and organization of the test cases. Notable testing frameworks for Solidity include Truffle, Hardhat [20], Waffle[21], and Remix[22]. Each framework comes with its unique features, merits, and drawbacks, necessitating a comparative analysis to determine which one aligns most effectively with your needs. In this proposal, we initially opted for Remix due to its user-friendly interface and simplicity. Subsequently, we transitioned to using Hardhat for more advanced testing in the later stages of development.

4. Unit Test: Unit tests consist of compact, small code segments that assess the performance of an individual function or contract. These tests are crucial for confirming that the code operates as intended and is free from errors or bugs. It is advisable to create unit tests for each function and contract, encompassing all conceivable scenarios and edge cases.
5. Integration Test: Integration tests employ more extensive and intricate code segments that assess the interplay among multiple functions or contracts. These tests are valuable for examining the logic and progression of the code, as well as its response to various inputs and outputs. It is recommended to develop integration tests for each feature or module introduced, simulating real-world conditions and events that code may encounter. Tools such as Ganache, Hardhat Network, or Forks can be utilized to establish a local blockchain environment tailored for the integration tests.
6. Code coverage: Employing code coverage tools is essential to ensure that the tests adequately cover all relevant aspects of the code and uphold effectiveness and reliability. Solidity-Coverage, Istanbul, and Hardhat are among the widely used code coverage tools for Solidity.
7. Gas Measuring: Calculate the gas consumed in deploying the contracts.

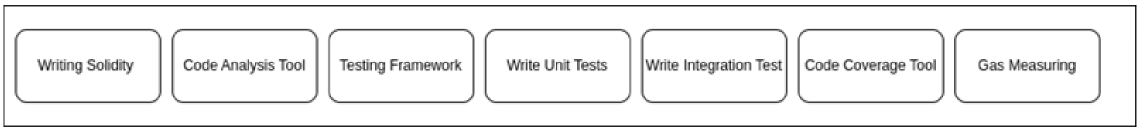


FIGURE: 6 Testing - Analysis Steps

5. MITIGATING SECURITY RISKS: PROPOSED SOLUTION

MYTHRIL [23] which is a dynamic analysis tool for Ethereum Virtual Machine (EVM) bytecode, is used to detect security issues and vulnerabilities.

5.1 Frontrunning SWC-114

This issue arises from the execution of e-will function, when generating one of the beneficiaries as a miner node in the blockchain. This node can disrupt the execution of the operation by raising the gas price for its transaction. The attacker can force his money transfer before the rest of the beneficiaries. To overcome the frontrunning problem, the system is redesigned by establishing a priority level for each beneficiary, ensuring that it is not dependent on the gas amount, an attacker can't achieve a transaction with a lower priority. The priority level is determined by the testator during writing the will.

5.2 Denial of Service SWC-128- Unexpected Revert

When attempting to send Ethers to multiple recipients with one function call, there is a potential problem where if one of the transfers fails e.g., wrong beneficiary address, all transfers that have already occurred are reverted and the following transfers will not execute. To address this issue, a technique is employed to isolate each external call, effectively transferring the risk of failure from the contract to the user. This method involves isolating the external transfer, so no other transfers rely on the successful execution of the isolated transfers. This technique is known as the Pull- over-Push pattern [24]. A map was created for refunds where each beneficiary's address grants him the right to withdraw his share following the testator's passing.

5.3 Reentrancy Problem SWC-107

After applying a push-over pull, another problem arises. The attacker gains control of the contract by recalling the withdraw function many times. Since the user’s balance is not set to 0 until the end of the function, each invocation may succeed and withdraw the balance repeatedly, even if one of the transfers fails. To prevent this issue, the share is cleared, followed by the transfer process. Additionally, a MUTEX is implemented to prevent an attacker from attempting to call the same function while the first call is still in progress. The lock prevents it from having any effect. In 2016, The DAO, a Decentralized Autonomous Organization was hacked, and 3.6 million Ether (approximately $50 million) was stolen using the first reentrancy attack. The Ethereum foundation issued a critical update to roll back the hack, which resulted in Ethereum being forked into Ethereum Classic and Ethereum.

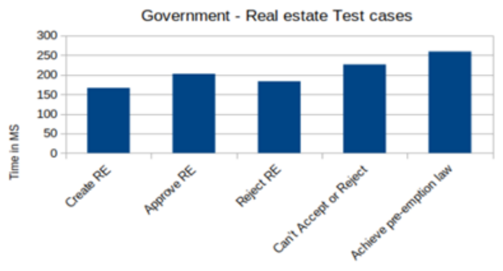
5.4 Self Destruct swc-106

The self-destruct is still running in many solidity versions. However, its use is highly discouraged because it eventually changes its semantics, and all contracts using it will be affected in some way. Self-destruct has been deprecated and warned against its use. In the proposed framework, the 'self-destruct' function is replaced with a new state called 'reject' which causes the contract to be invalid.

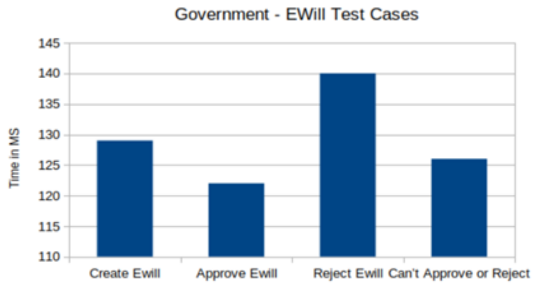
6. EXPERIMENTAL RESULTS AND DISCUSSION

Hardhat is utilized for debugging the framework, as indicated in [26]. Eighteen test cases, designed to simulate real-world framework usage, were created to ensure comprehensive testing and alignment with user requirements.

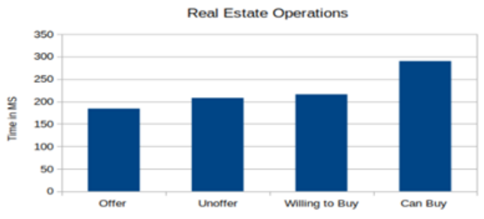
The first group tests are to 'create real estate'. This function tests that any citizen can send a request for a government contract to request ownership approval for his property. 'Approve RE' and 'Reject RE' functions tested to ensure the restriction rules. 'Achieve preemption law' function is also tested. Figure 7a shows the average time consumed for testing all the functions which include interactions between government contract and real estate contract. The graph shows that the maximum time consumed for the government to verify the preemption law is around 250ms. The second group of four test cases is for the functions that include interaction between the government and the ewill contract. 'Create Ewill' tests the behavior of the contract after sending many created ewill requests. All the cases passed with time below 150 MS, as shown in figure 7b The functions related to real estate contract are tested: 'Offer', 'Unoffer', 'Willing to buy', and 'canBuy' these functions are internal in real estate contract and don't need government approval as depicted in figure 7c. Finally, the functions related to the ewill contract are tested to show the time consumed for running the following operations: 'Deposit' and 'Withdraw' both functions successfully allowed the owner to add and withdraw money from his contract. 'Add money beneficiary' succeeds in designating someone who can receive funds from the contract. 'Add real estate entry' function succeeded to add information about a real estate property to the contract. This function also succeeded to transfer the ownership to the beneficiary after the testator's death by inheritance. All relevant data are recorded, and the execution time for all the aforementioned operations is measured and recorded in figure 7d.



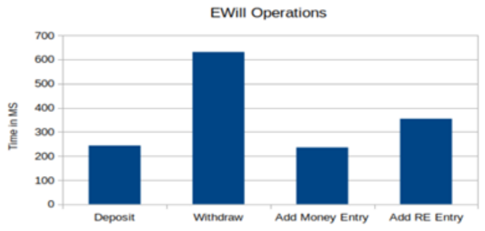
a) Government and Real Estate Interaction



b) Government and E-will Interaction



C) Real Estate Operations



d) E-will Operations

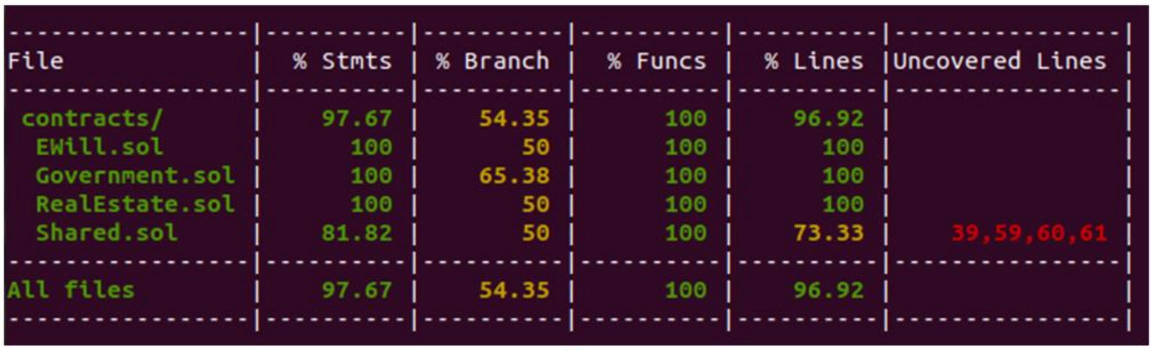
FIGURE 7: Timing measurements for all the functions within the framework

6.1 Measuring Code Coverage

Code coverage is a metric that can help in understanding how much of the source code is tested. It guarantees that the smart contract functions are tested with all possible outcomes and scenarios. The common metrics mentioned in the coverage reports are: 1- Function coverage, the main metric that measures how many of the functions defined within the contracts have been called. 2- Statement coverage that test ensures that each executable statement within the source code is run at least once. 3- Branches coverage assesses the execution of branches within control structures, such as if statements, to determine the coverage achieved during testing. 4- Line coverage measures the proportion of source code lines that have been tested. Typically, these four metrics are presented in terms of the number of tested items, the total items present in the code, and a coverage percentage calculated as the ratio of tested items to total items found [18]. The test results demonstrate a 100% coverage of all functions tested. The only exception is the 'shared' file, the shared file contains the shared variables among all contracts. Figure 8 represents the coverage percentage of our test.

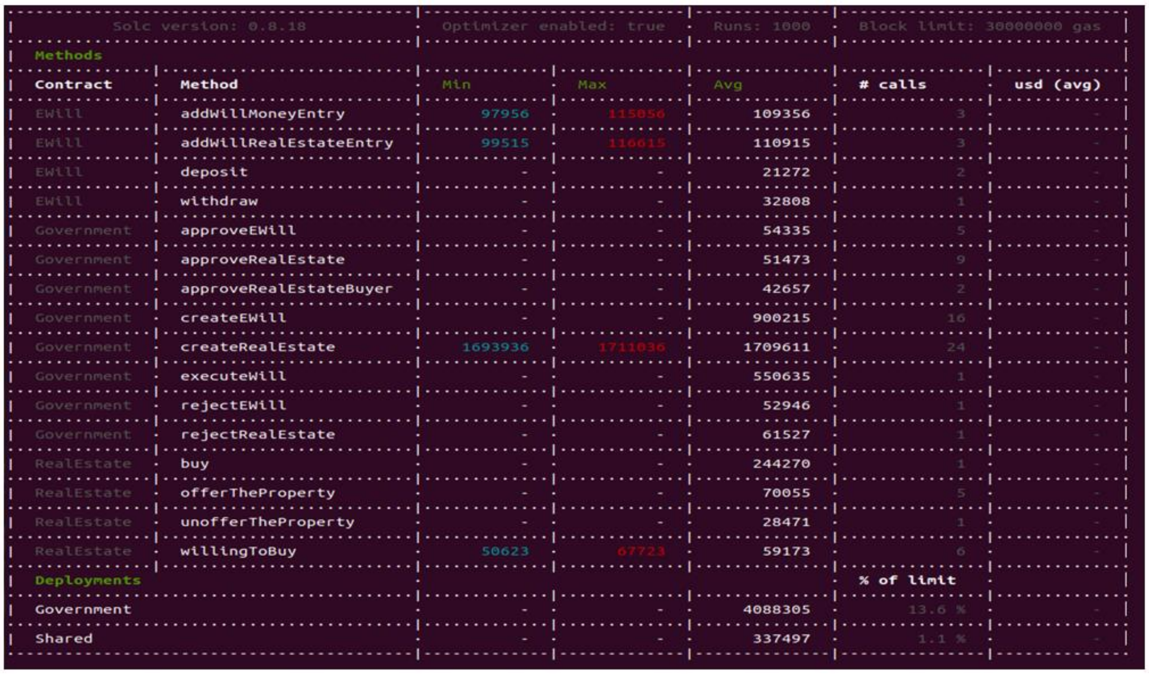
6.2 Measuring Gas Consumption

A very important metric is gas consumption which represents a metric used to gauge the computational workload of specific operations. The greater the complexity of an operation, the higher the gas consumption [25]. Average gas usage per method is also reported using 'Hardhat' plugin tool to get metrics of how much gas is used [18], based on the execution of the previous tests. Figure 9 shows the gas consumption records. The report shows the average gas cost for each function and the deployment cost for the main contract. The contract is compiled using an 'optimizer', a built-in function, which reduces either the total bytecode size or the gas required to execute certain functions. The functions that don't have min-max reports don't have optimization suggestions. The block limit indicates the maximum number of gas units that can be accommodated within a single block. Various networks may have distinct values for this limit, with some featuring dynamically adaptable constraints. Moreover, in emulators or private networks, it's often possible to customize and set one's limit.



```
-----------------|----------|----------|----------|----------|----------------|
File             |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------|----------|----------|----------|----------|----------------|
 contracts/      |    97.67 |    54.35 |      100 |    96.92 |                |
  EWill.sol      |      100 |       50 |      100 |      100 |                |
  Government.sol |      100 |    65.38 |      100 |      100 |                |
  RealEstate.sol |      100 |       50 |      100 |      100 |                |
  Shared.sol     |    81.82 |       50 |      100 |    73.33 |    39,59,60,61 |
-----------------|----------|----------|----------|----------|----------------|
All files        |    97.67 |    54.35 |      100 |    96.92 |                |
-----------------|----------|----------|----------|----------|----------------|
```

FIGURE 8: Code Coverage Test



```
------------------------------------------------|---------------------------|-------------|-----------------------------
|         Solc version: 0.8.18                  ·  Optimizer enabled: true  ·  Runs: 1000 ·  Block limit: 30000000 gas  |
················································|···························|·············|······························
|  Methods                                                                                                               |
··············|·································|·············|·············|·············|···············|··············
|  Contract   ·  Method                         ·  Min        ·  Max        ·  Avg        ·  # calls      ·  usd (avg)  |
··············|·································|·············|·············|·············|···············|··············
|  EWill      ·  addWillMoneyEntry              ·      97956  ·     115056  ·     109356  ·            3  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  EWill      ·  addWillRealEstateEntry         ·      99515  ·     116615  ·     110915  ·            3  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  EWill      ·  deposit                        ·          -  ·          -  ·      21272  ·            2  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  EWill      ·  withdraw                       ·          -  ·          -  ·      32808  ·            1  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  approveEWill                   ·          -  ·          -  ·      54335  ·            5  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  approveRealEstate              ·          -  ·          -  ·      51473  ·            9  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  approveRealEstateBuyer         ·          -  ·          -  ·      42657  ·            2  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  createEWill                    ·          -  ·          -  ·     900215  ·           16  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  createRealEstate               ·    1693936  ·    1711036  ·    1709611  ·           24  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  executeWill                    ·          -  ·          -  ·     550635  ·            1  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  rejectEWill                    ·          -  ·          -  ·      52946  ·            1  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Government ·  rejectRealEstate               ·          -  ·          -  ·      61527  ·            1  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  RealEstate ·  buy                            ·          -  ·          -  ·     244270  ·            1  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  RealEstate ·  offerTheProperty               ·          -  ·          -  ·      70055  ·            5  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  RealEstate ·  unofferTheProperty             ·          -  ·          -  ·      28471  ·            1  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  RealEstate ·  willingToBuy                   ·      50623  ·      67723  ·      59173  ·            6  ·          -  |
··············|·································|·············|·············|·············|···············|··············
|  Deployments                                  ·                                         ·  % of limit   ·             |
················································|·············|·············|·············|···············|··············
|  Government                                   ·          -  ·          -  ·    4088305  ·       13.6 %  ·          -  |
················································|·············|·············|·············|···············|··············
|  Shared                                       ·          -  ·          -  ·     337497  ·        1.1 %  ·          -  |
------------------------------------------------|-------------|-------------|-------------|---------------|-------------
```

FIGURE 9: Gas Consumption for each function in the framework

7. CONCLUSION

Blockchain changes how government functions, paving the way for new service delivery models. Governments have expressed interest in using blockchain technology, However, the technology's application and use cases for offering public services are facing vagueness since there aren't many published studies, and standardized models that are well accepted and can be used to make comparisons. This research intensively investigates successful instances where blockchain technology has been applied to introduce innovative services in various countries. The study then focuses on studying the problems for the current system in the Egyptian government and introduces the first proposed framework for the Egyptian government's G2C service to facilitate the adoption of blockchain technology in two related services, the real estate sector, and e-will. Detailed sequence diagrams are designed that Identify the main actors involved and their roles within the model. The system is implemented using solidity and deployed first using remix IDE and then deployed using vscode, node js v16.17.1, solidity 0.8.9, and hardhat tools. The Mythril is then used for testing the vulnerabilities that exist in the smart contract to evaluate the effectiveness and feasibility of our proposed framework, a series of experiments is carried out utilizing typical real estate scenarios. However, our attempts to carry out benchmarking were hindered by the absence of any quantitative results and the lack of published specifications. Future work will focus on expanding the model horizontally to include more services in Egypt. Additionally, using smart contracts to build a homogeneous blockchain that can communicate with one another, the ability to distribute applications, and smart contracts between other blockchain networks is a useful case study for future work.

8. References

[1] Kerkhoff, D.A.D.N.: Blockchain, good governance and public procurement. In: LeidenUniversity, Faculty of Governance and Global Affairs (2021)

[2] Nakamoto, Satoshi.” Bitcoin: A peer-to-peer electronic cash system.” Decentralized business review (2008).

[3] Lu, H., Huang, K., Azimi, M., Guo, Blockchain technology in the oil and gas industry: A review of applications, opportunities, challenges, and risks. Ieee Access 7,41426–41444 (2019)

[4] Zou, W., Lo, D., Kochhar, P.S., Le, X.-B.D., Xia, X., Feng, Y., Chen, Z., Xu, B.:Smart contract development: Challenges and opportunities.

IEEE Transactions on Software Engineering 47(10), 2084–2106 (2019).

[5] Mourtzis, D., Angelopoulos, J., Panopoulos, N.: Blockchain integration in the era of industrial metaverse. Applied Sciences 13(3), 1353(2023).

[6] Comincioli, L.M.: The role of blockchain in improving land-users' rights (can blockchain solve corruption problems in land administration in developing countries? - the case of India). (July 2021).

[7] Chu, H., Zhang, P., Dong, H., Xiao, Y., Ji, S.,Li, W.: A survey on smart contract vulnerabilities: Data sources, detection and repair.Information and Software Technology, 107221(2023).

[8] He, D., Wu, R., Li, X., Chan, S., Guizani,M.: Detection of vulnerabilities of blockchain smart contracts. IEEE Internet of Things Journal (2023).

[9] Mense, A., Flatscher, M.: Security vulnerabilities in Ethereum smart contracts. In: Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services, pp. 375–380 (2018).

[10] C Liu, Z.C.Z.C.B.C. H Liu: Reguard: finding reentrancy bugs in smart contracts. In: International Conference on Software Engineering, pp. 65–68 (2018).

[11] Daian, P., Goldfeder, S., Kell, T., Li, Y.,Zhao, X., Bentov, I.Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. arXiv preprintarXiv:1904.05234 (2019)).

[12] https://en.eipss-eg.org/egypt-real-estate-registration-crisis-policies-scenarios / (2021)

[13] https://mnasserlaw.com/real-estate-registration-law / (2024)

[14] Elsheryie, K.A., 2023. Reformation of Will Statutes and the Legality of eWills.

[15] https://www.nytimes.com/2021/01/13/business/tens-of-billions-worth-of-bitcoin-have-been-locked-by-people-who-forgot-their-key.html

[16] Solidity 0.8.9. https://github.com/ethereum/solidity/releases/tag/v0.8.9. (2021)

[17] Palechor, L., Bezemer, C.-P.: How are solidity smart contracts tested in open source projects? an exploratory study. In: Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test,pp. 165–169 (2022).

[18] HardHat https://hardhat.org/.(2024)

[19] Manal Mansour , May Salama , Hala Helmi , Mona F.M Mursi.: A Survey on Blockchain in E-Government Services: Status and Challenges. International Journal of Engineering Research ans Technology (IJERT) Volume 12, Issue 4 (2023).

[20] Jain, S.M.: Hardhat. In: A Brief Introduction to Web3: Decentralized Web Funda mentals for App Development, pp. 167–179.Springer, (2022).

[21] Paulavicius, R., Grigaitis, S., Filatovas, E.:A systematic review and empirical analysis of blockchain simulators. IEEE access9,38010–38028 (2021).

[22] Amir Latif, R.M., Hussain, K., Jhanjhi, N.,Nayyar, A., Rizwan, O.: A remix ide: smart contract-based framework for the healthcare sector by using blockchain technology. Multimedia tools and applications, 1–24 (2020).

[23] MythX https://github.com/ConsenSys/mythril.(2023)

[24] Pulloverpush https://fravoll.github.io/solidity-patterns/pull_over_push.html (2023)

[25] Farokhnia, S., Goharshady, A.K.: Reducing the gas usage of ethereum smart contracts without a sidechain. In: 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–3 (2023).

[26] Modi, R.. Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and blockchain. Packt Publishing Ltd (2018).