JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

IMPERATIVE PROCEDURE OF PROCESSING A TRANSACTION IN SMART CONTRACT

T Anne Sanjana **UVS Tejaswani** **U Meena** **C Narasimham (IOCLChairprofessor)**

Abstract:

This study explores the imperative implementation of smart contracts within the decentralized technologies, offering a comprehensive analysis of their design and execution. With autonomous and self-executing characteristics, smart contracts play a vital role in securing transactions in block chain network. This research work is proposed to provide clear instructions in defining smart contracts and deploying them efficiently. Furthermore this study proposes the concept of token generation and sharing through object serialization. This work aims to indicate the successful deployment of smart contracts for transparent transaction in the block chain network or platform (say, Ethereum).

Introduction:

In the rapidly evolving environment of decentralized technologies, smart contracts have emerged as a breakthrough innovation that facilitates secure, transparent and autonomous transactions within block chain networks. Smart contracts are digital contracts that execute automatically when specific conditions are met. They run on the block chain and ensure that transactions take place exactly as programmed without the need for an intermediary. They conduct secure and transparent transactions. An imperative smart contract is a type of self-executing digital contract on the block chain that follows an imperative programming approach. It specifies a sequential sequence of actions to be taken when predefined conditions are met and provides a clear and detailed set of instructions for performing the contract. Smart contracts are also called cutting-edge technology in block chain, which are developed to negotiate agreements between two or more parties in a decentralized setting. As block chain technology has evolved, smart contracts have proven to be more efficient than traditional contracts, which are more susceptible to greater risks. Smart contracts have certain parameters that are passed to the code and affect its behavior or execution. The main key components are variables, functions, timestamps, block numbers, events, and gas parameters. If you want to necessarily develop code from logic, there are several platforms used by smart contracts. These platforms provide different consensus methods and programming languages for the contract. While this work uses the Ethereum platform to write smart contracts in a specific language called Solidity, which is the most popular, there are others such as Hyper ledger Fabric which uses Java, C++, NodeJs, etc. For imperative (also called procedural) smart contracts must be programmed in imperative or declarative languages.

Problem Analysis:

Develop an efficient and secure imperative process for smart contract transaction processing that ensures accuracy, transparency, and reliability while minimizing execution complexity risks and maintaining consistency across decentralized networks. It manages multiple transaction inputs, executes complex business logic, and maintains consistency across a distributed ledger. The goal is to design and implement an imperative approach for smart contract development that addresses the challenges of complex transaction processing, security vulnerabilities, gas optimization, scalability, and adherence to best practices. Automation is one of the biggest advantages that smart contracts provide over traditional contracts as self-executing code that automatically enforces the terms of the contract when predefined conditions are met. The smart contracts efficiently deals with legal and compliance issues and ensures that smart contract complies with relevant legal requirements and consider legal review when dealing wish sensitive transactions and smart contracts should use clear and well documented code and consider adopting more transparent paradigms for solving lack of transparency.

Design of The System:

a)Token Generation:

A token is a digital asset that represents ownership or access rights. While the term "token" can be used in various contexts in computer science and cryptography, in the block chain space it specifically refers to units of value on the block chain.

b) Token Sharing:

Token sharing involves the exchange or distribution of a cryptographic key or token between two parties in a secure and controlled manner. These tokens are often used to facilitate secure transactions, access decentralized applications, or represent ownership of assets on the block chain. The token can be generated and shared in two ways: 1) each of the two parties involved can generate a token and then share it with the other party.2) A party with a higher priority can create a token and distribute it to the other party. Any third party can generate a token and share it with both parties involved. The token can be generated using any of the above methods.

c) Object Serialization:

Serialization is the process of converting the state of an object into a format that can be easily stored or transferred and later reconstructed. Simply put, it is the conversion of an object into a byte stream that can be saved as a memory file. This file can be de-serialized to get the original object.

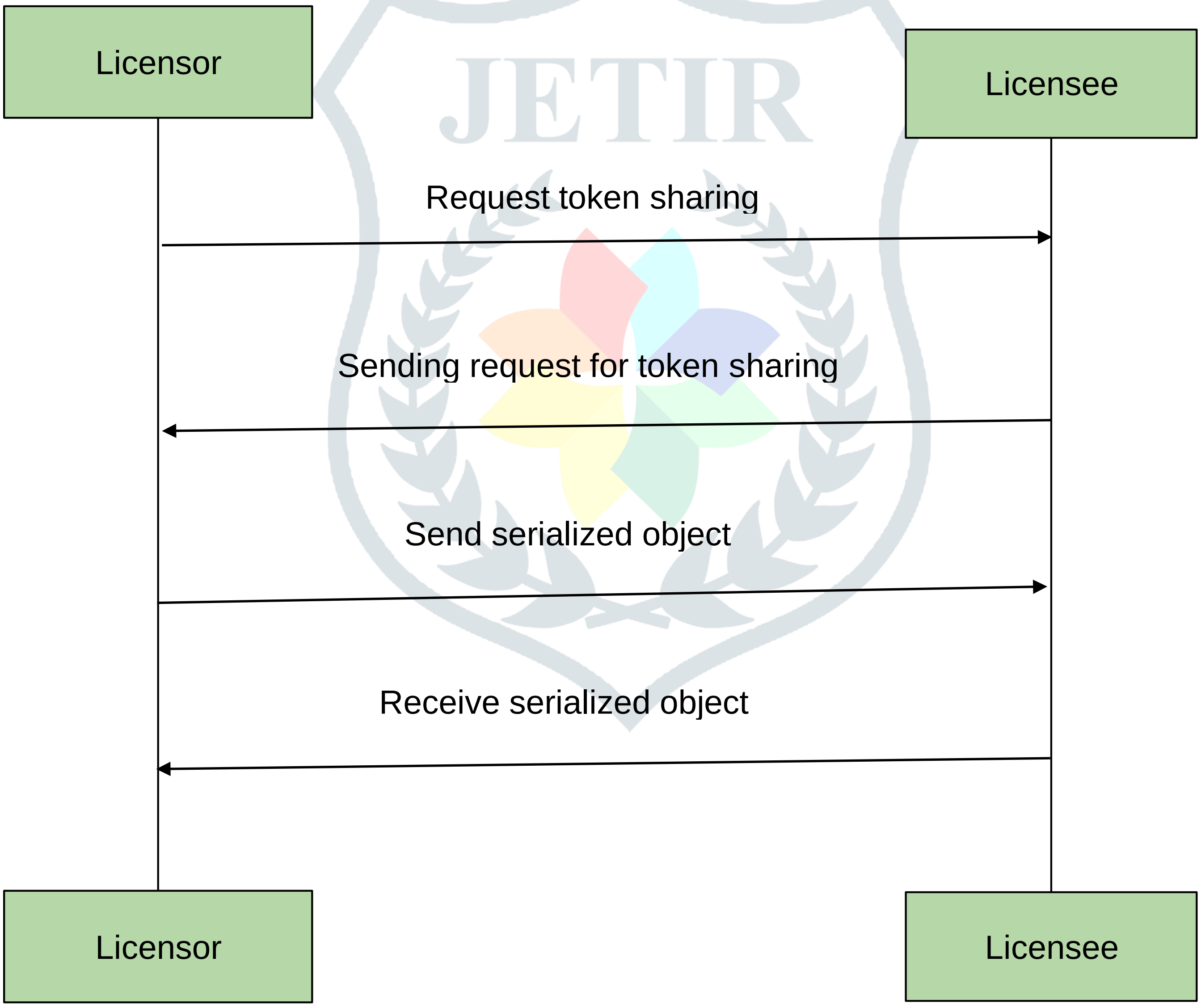


Fig-01: Serialization of objects between provider and licensee

d) Establishment Of Smart Contract:

Smart contracts work by converting business code into logic and running on a virtual machine connected to the block chain network. The creation of a smart contract begins with an agreement between the parties, i.e. the "Licensor" and the "Licensee". Then the terms of the contract are translated into a programming language to create a smart contract. It specifies the rules just like a traditional legal contract. A smart contract is automatically executed when the conditions in the logic are met.

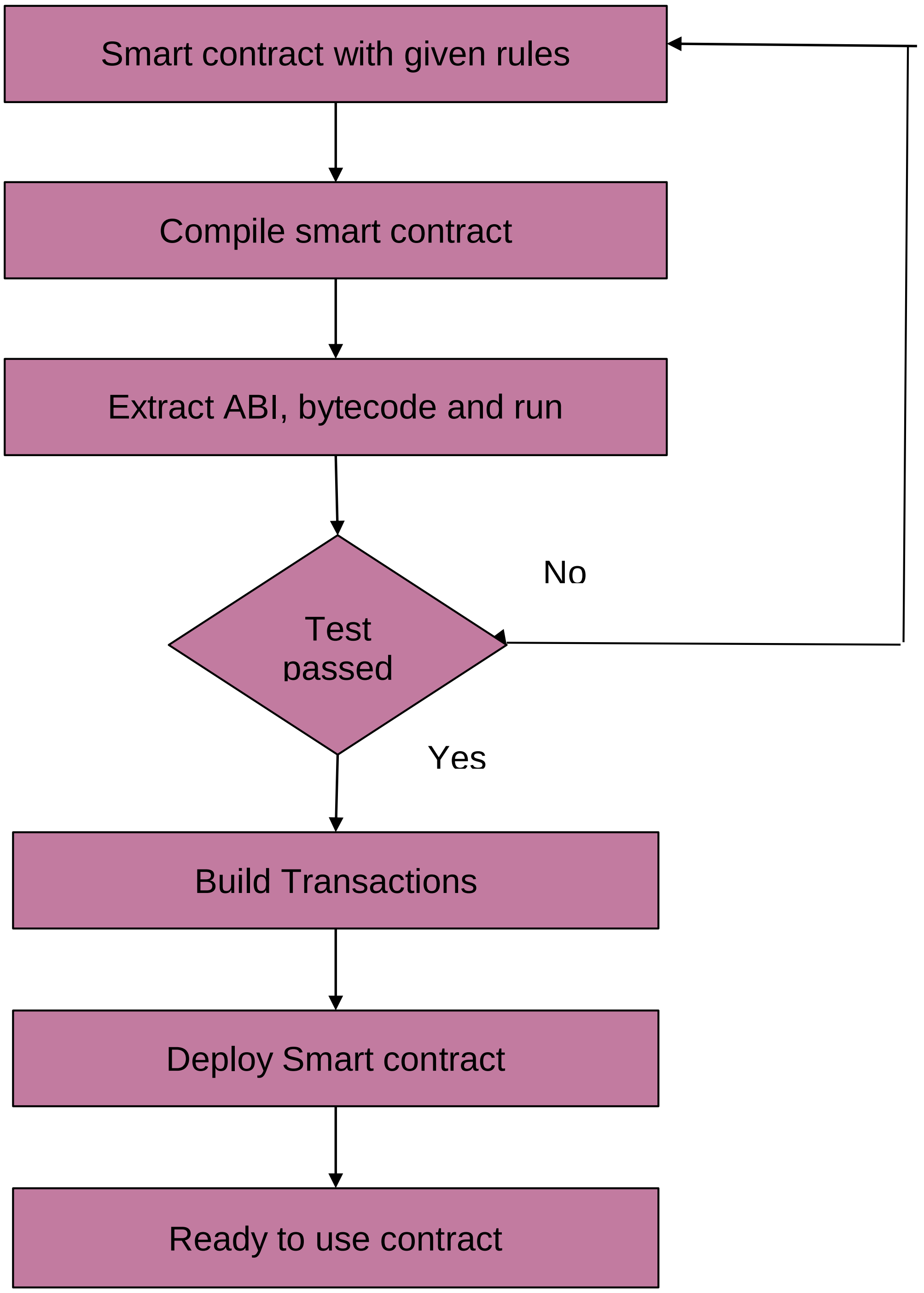


Fig-02: Smart contract workflow

e) Deployment:

There are several ways to deploy a smart contract depending on the block chain platform. The work uses the Ethereum block chain to develop and deploy a smart contract. The smart contract is deployed using "Ganache". Deployment involves sending an Ethereum transaction containing the compiled smart contract code.

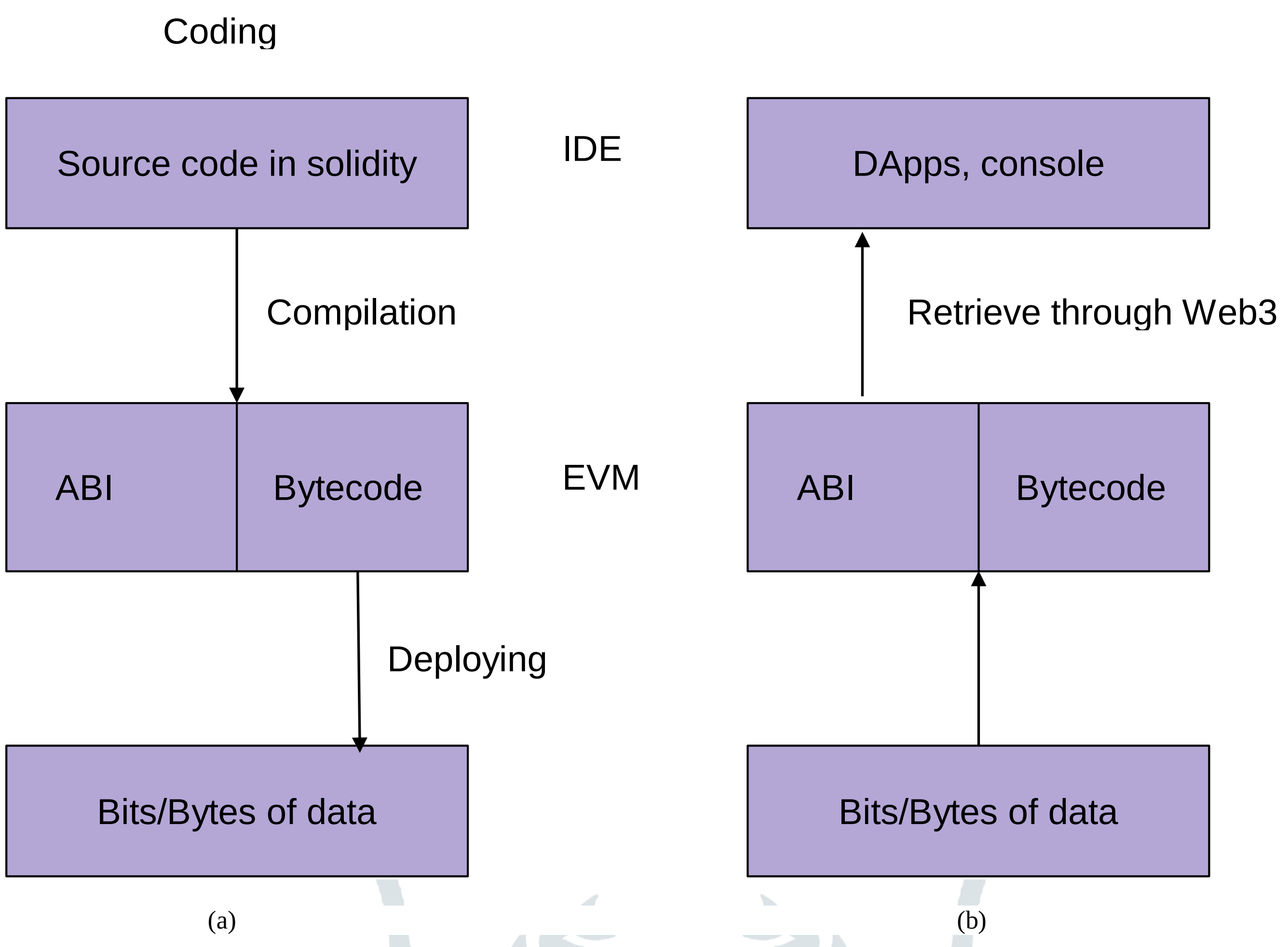


Fig-03: (a) Writing smart contract to Ethereum, (b) Reading contract from Ethereum

Implementation:

a) Token Generation:

The token can be generated in Python. In this Python script, a secure token generation feature is implemented using the Hashlib library. The code hashes the randomly generated token using the SHA-256 algorithm. The generated token can be used for authentication or transaction verification, providing a robust mechanism for generating cryptographic tokens in applications requiring data integrity and time relevance.

Findings:

173561e93b22f2bc467a305126e434b4143955ec5cd65220e7bd30eb72c29c88

b) Token Sharing (Object Serialization):

The code provided implements the basic mechanism for sharing tokens in the context of a smart contract. The `Token` class facilitates serialization and deserialization, allowing tokens to be shared between entities. The token generated by the "Licensor" class contains a value and an expiration time. The "Licenser" class creates a token with a randomly generated amount and expiration time. The licensor generates secure tokens using the token generation method and shares the generated tokens with the licensee.

Findings:

Received Token Value:

91bc496b206a9fad6daba8e1b64b0f906237b12636383320bec29d78384789a6

Received Token Expiration Time: 2024-01-21 05:56:03.834705

c) Smart Contract In Solidity:

The terms and conditions for evaluation of an agreement or a contract are written inside the code. This code is written in solidity on the Ethereum platform. It describes the terms for license validation, violation, and termination. The contract is then compiled.

Findings:

Retrieving compiler information:

Compiler using remote version: 'v0.8.19+commit.7dd6d404', solidity version: 0.8.19+commit.7dd6d404.Emscripten.clang

Compilation completed successfully!

d) Read The Solidity File In Python:

After successful compilation the contract is read in python using the path of the solidity file. For establishing the smart contract on python web3 is an important tool. The 'load_dotenv' is used for managing configuration and securing sensitive information like URLs, API keys.

e) Compile Solidity Smart Contract In Python:

The smart contract's source code is read and compiled. Make sure that the solidity compiler used is compatible with the version of the one installed. The contract is then serialized into a json formatted string using 'json.dump'.

```
compiled_sol = compile_standard( {
      "language": "Solidity",
      "sources": {"License.sol": {"content": simple_storage_file}},
      "settings": {
         "outputSelection": {
            "*": {
               "*": ["abi", "metadata", "evm.bytecode", "evm.bytecode.sourceMap"]
            }         }       }    }
   solc_version="0.8.19", )
with open("compiled_code.json", "w") as file:
   json.dump(compiled_sol, file)
```

f) Extract The ABI And Bytecode:

The below code extracts abi and bytecode. This work includes smart contracts written in python which is a high level language. Inorder to run the smart contract it should be compiled into EVM bytecode. The abi acts as an intermediary between the bytecode and the python code to interact with each other.

```
abi = json.loads(compiled_sol["contracts"]["License.sol"]["License"]["metadata"]  )["output"]["abi"]
 bytecode =compiled_sol["contracts"]["License.sol"]["License"]["evm"]["bytecode"]["object"]
```

g) Setup Connection To Local Ethereum Blockchain(Ganache):

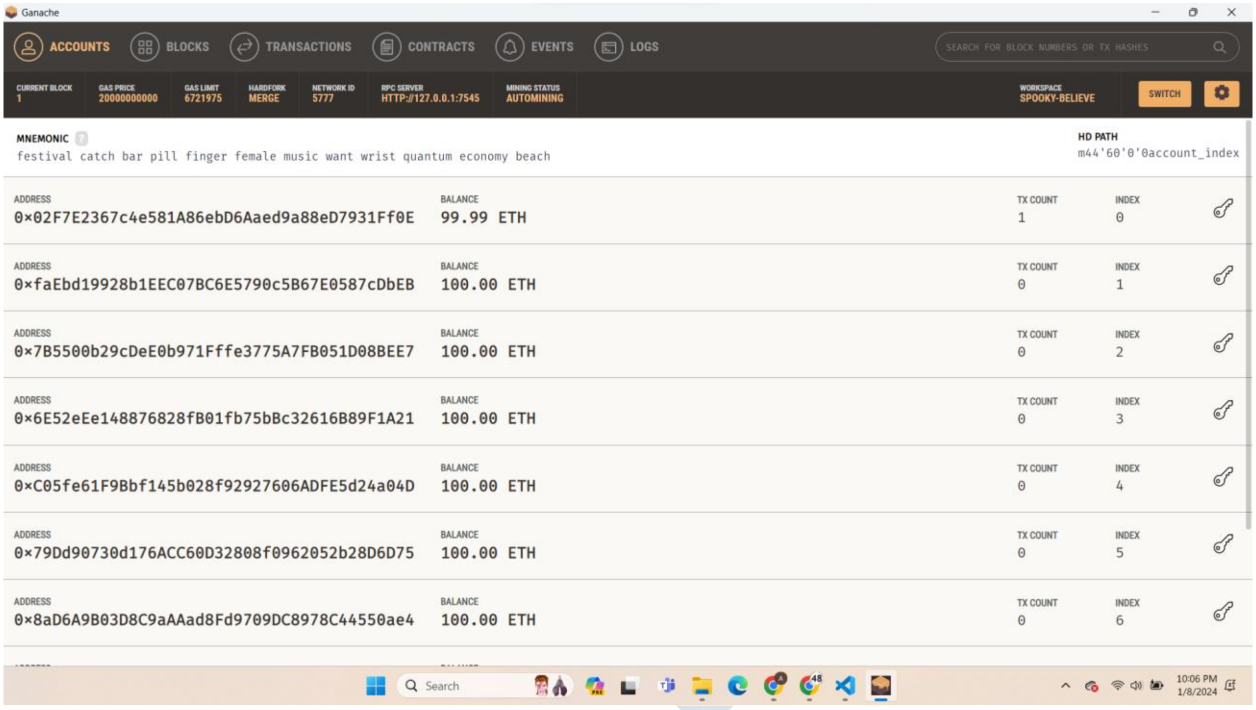
To establish connection, an HTTP web address(of Ganache) is provided. The address and the private key of the sender is specified in code.

```
w3 = Web3(Web3.HTTPProvider("HTTP://127.0.0.1:7545"))
chain_id = 1337
my_address = "0x02F7E2367c4e581A86ebD6Aaed9a88eD7931Ff0E"
private_key= "0xc8fa88b9cba7e0582dbeb9d2182250cb1148b30249ca72493e32d822f18e03ab"
```

h) Build Transaction:

```
License = w3.eth.contract(abi=abi, bytecode=bytecode)
nonce = w3.eth.get_transaction_count(my_address)
tx = License.constructor().build_transaction(  {
        "chainId": chain_id,
        "gasPrice": w3.eth.gas_price,
        "from": my_address,   "nonce": nonce,   })
```

Transaction In Ganache:



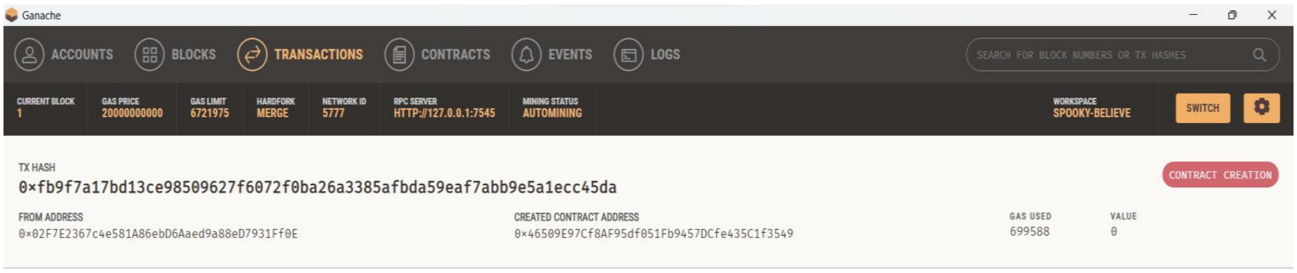
```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK  NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
1              20000000000  6721975    MERGE     5777        HTTP://127.0.0.1:7545  AUTOMINING      SPOOKY-BELIEVE

MNEMONIC                                                                            HD PATH
festival catch bar pill finger female music want wrist quantum economy beach        m44'60'0'0account_index

ADDRESS                                     BALANCE     TX COUNT  INDEX
0×02F7E2367c4e581A86ebD6Aaed9a88eD7931Ff0E  99.99 ETH   1         0
0×faEbd19928b1EEC07BC6E5790c5B67E0587cDbEB  100.00 ETH  0         1
0×7B5500b29cDeE0b971Fffe3775A7FB051D08BEE7  100.00 ETH  0         2
0×6E52eEe148876828fB01fb75bBc32616B89F1A21  100.00 ETH  0         3
0×C05fe61F9Bbf145b028f92927606ADFE5d24a04D  100.00 ETH  0         4
0×79Dd90730d176ACC60D32808f0962052b28D6D75  100.00 ETH  0         5
0×8aD6A9B03D8C9aAAad8Fd9709DC8978C44550ae4  100.00 ETH  0         6
```

i) Deployment of Smart Contract:

```
signed_tx = w3.eth.account.sign_transaction(tx, private_key=private_key)
tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
print(f"Contract deployed to {tx_receipt.contractAddress}")
```

Contract Deployment in Ganache:

After a successful deployment, the contract creation and transaction looks like this:



CURRENT BLOCK 1 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MERGE | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE SPOOKY-BELIEVE

TX HASH
0×fb9f7a17bd13ce98509627f6072f0ba26a3385afbda59eaf7abb9e5a1ecc45da
CONTRACT CREATION

FROM ADDRESS
0×02F7E2367c4e581A86ebD6Aaed9a88eD7931Ff0E

CREATED CONTRACT ADDRESS
0×46509E97Cf8AF95df051Fb9457DCfe435C1f3549

GAS USED
699588

VALUE
0

Conclusion:

The work outlines a detailed procedure of establishing smart contracts and constructing a transaction in block chain technology using both imperative and declarative features. Further, the study explores the deployment of smart contracts with ease, leveraging various libraries and methods. Besides direct implementation of the contract, this research work mentions the concept of token sharing through object serialization giving insights into object oriented approach to processing a transaction. The findings indicate that transactions within smart contracts exhibit enhanced security than traditional contracts with fulfillment of all the legal objectives.

References:

[1] Hamed Taherdoost, *Smart Contracts in Blockchain Technology: A Critical Review*, University Canada West.

[2] Guido Governatori, Florian Idelberger, Zoran Milosevic, Regis Riveret, Giovanni Sartor, Xiwei Xu *On legal contracts, imperative and declarative smart contracts, and blockchain systems*, CSIRO, Australia, European University Institute, Italy

[3] Sara Rouhani, Ralph Deters *Security, Performance, and Applications of Smart Contracts: A Systematic Survey* Department of Computer Science, University of Saskatchewan, Saskatoon, SK, Canada.Published by IEEE.

[4] P. L. Seijas, S. J. Thompson and D. McAdams, *Scripting smart contracts for distributed ledger technology*, 2016.

[5] Jeff Daniel, Arman Sargolzaei, Mohammed Abdelghani, Saman Sargolzaei, Ben Amaba, *Blockchain Technology, Cognitive Computing, and Healthcare Innovations.*

[6] N. Prusty, *Building blockchain projects: develop real-time practical DApps using Ethereum and JavaScript.* Livery Place, 35 Livery Street, Birmingham, B3 2PB, UK: Packt Publishing Ltd., ISBN 978-1-78712-214-7, 2017.

[7] Deploying smart contracts. https://ethereum.org/en/developers/docs/smart-contracts/deploying.

[8] G. Governatori. *Representing business contracts in RuleML. Int. Journal of Cooperative Information Systems,* 14(2-3):181–216, June-September 2005.

[9] G. Governatori and D. H. Pham. DR-CONTRACT: An architecture for e-contracts in defeasible logic. *Int. Journal of Business Process Integration and Management,* 5(4), May 2009.

[10] Smart contracts in blockchain technology and how they work https://www.geeksforgeeks.org/smart-contracts-in-blockchain/.