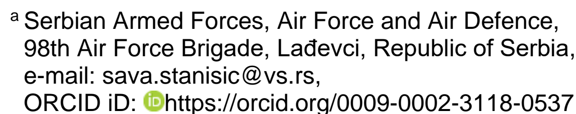
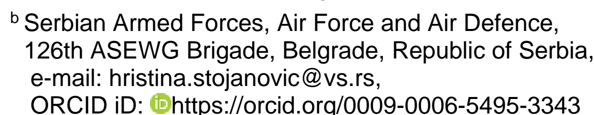
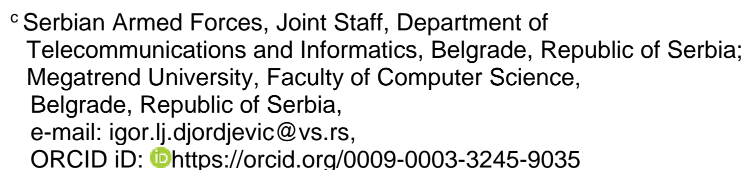
The utilization of Solidity programming language in blockchain

Sava S. Stanišić[a], *Hristina* N. Stojanović[b], *Igor* Lj. Đorđević[c]


[a] Serbian Armed Forces, Air Force and Air Defence,
98th Air Force Brigade, Lađevci, Republic of Serbia,
e-mail: sava.stanisic@vs.rs,
ORCID iD: https://orcid.org/0009-0002-3118-0537


[b] Serbian Armed Forces, Air Force and Air Defence,
126th ASEWG Brigade, Belgrade, Republic of Serbia,
e-mail: hristina.stojanovic@vs.rs,
ORCID iD: https://orcid.org/0009-0006-5495-3343


[c] Serbian Armed Forces, Joint Staff, Department of Telecommunications and Informatics, Belgrade, Republic of Serbia;
Megatrend University, Faculty of Computer Science,
Belgrade, Republic of Serbia,
e-mail: igor.lj.djordjevic@vs.rs,
ORCID iD: https://orcid.org/0009-0003-3245-9035

DOI: https://doi.org/10.5937/vojtehg72-47942

FIELD: computer sciences, IT
ARTICLE TYPE: review paper

Abstract:

Introduction/purpose: This work provides a comprehensive overview of blockchain technology, elucidating its foundational principles and how it ensures transparency, immutability, and decentralization. The integration of Solidity with blockchain is explored through theoretical approach.

Methods: This work meticulously dissects blockchain principles, elucidating transparency, immutability, and decentralization, while exploring Solidity integration in a theoretical framework, ensuring a comprehensive understanding of their intricate relationship and contributing to a broader comprehension of modern distributed ledger technology.

Results: The resulting product of this paper will be getting useful knowledge about the technology that practically shapes the world.

Conclusion: In conclusion, the adoption of Solidity as a programming language in blockchain technology has proven to be pivotal, enhancing smart contract functionality and overall system security. Its specialized features make it an indispensable tool for developers navigating the complexities of decentralized applications.

Key words: blockchain, Bitcoin, Ethereum, Solidity, decentralization.

Introduction

In the continuously evolving landscape of blockchain technology, the emergence of decentralized applications (DApps) and smart contracts has become a focal point of innovation. At the epicenter of this transformative shift lies Solidity, a purpose-specific programming language meticulously engineered for the development of secure and autonomous smart contracts on blockchain networks, prominently exemplified by Ethereum. This scholarly inquiry seeks to systematically investigate the profound implications and nuanced intricacies surrounding the "Utilization of Solidity programming language in blockchain." Throughout this paper, the objective is to delineate the fundamental role Solidity assumes in concretizing the theoretical constructs of decentralized systems into practical, implementable solutions. Through the scrutiny of the syntax, structure, and unique attributes of Solidity, the aim is to provide a scholarly discourse that advances the understanding of the intricate interplay between this programming language and the broader landscape of blockchain technology.

Moreover, this scholarly effort endeavors to dissect the practical implications of Solidity-driven smart contracts, considering their impact across various sectors such as finance, supply chain, and governance. By elucidating the distinct features of Solidity that contribute to the resilience and immutability of smart contracts, the authors aim to provide an explanation of a framework for researchers, developers, and industry practitioners navigating the burgeoning field of decentralized applications. The exploration extends beyond the syntax and semantics of Solidity, encompassing considerations of security measures, standardization, and potential avenues for future enhancements. As the discourse unfolds, the anticipation is to shed light on the challenges inherent in the application of Solidity in real-world scenarios, thereby contributing valuable insights to the ongoing dialogue surrounding the convergence of Solidity and blockchain technology.

Blockchain basics

Understanding the principles on which blockchain resides is not possible without understanding the answers to the following questions:

- What is the purpose of blockchain? and
- Why is blockchain needed for cryptocurrency?

In simple terms, the purpose of blockchain is to have a network of computers agree upon a common state of data (Abou Jaoude & Saade,

2019). Any person or organisation should be able to participate in this process and no person or ogranisation should be able to control this process. This will be explained in the following chapters.

Also, blockchain solves the problem of **trust**. This system is completely neutral and resistant to any censorship or bribe. In 2008, an individual, or a group of people, under the pseudonym of Satoshi Nakamoto released a whitepaper for Bitcoin: "We have proposed a system for electronic transactions without relying on trust." (Nakamoto, 2008)

The system proposed was peer-to-peer network, allowing online payments to be directly sent from one party to another, without going through a financial institution, therefore eliminating the need for trusting someone to make sure your payment goes the way it is meant to.

The Genesis of blockchain

It is impossible to look back at the history of blockchain without looking back at the history of cryptography and its development over the years.

Until the 1970s, cryptography was the study of encrypting messages to the full decryption-proof stadium. It was used for passing confidential information, especially within the military (Ahmad et al, 2021). Substitution ciphers were primarily used – the cryptographical method of encrypting in which units of plaintext are replaced with ciphertext in a defined manner.

As cryptography advanced over the years, more and more complex functions were introduced. The most important leap for blockchain technology, certainly, was the idea of a secret key.

Namely, if two parties can meet prior to their exchange of messages and agree upon a common key for both sides, the message would be encrypted with a mathematical function and that key, creating even more secure encryption. This thus marked the beginning of symmetric-key cryptography.

With the advent of personal computing, cryptographers started to think even further. The idea of secure communication without prior key exchange emerged. In 1976, Whitfield Diffie proposed a concept of PKC (Public Key Cryptography). With PKC, each individual has their own unique key pair, consisting of a public key and a private key. Only the public key needs to be exchanged, eliminating the need for exchanging keys beforehand. If a person's public key is used to encrypt a message, then only their corresponding private key can decrypt it, providing privacy. Likewise, if their private key is used to sign (encrypt) a message, the corresponding public key can authenticate (decrypt) the message. This was the start of assymetric encryption.

Diffie did not have any practical way to make this happen. He had a concept, but the mathematical function with all these properties did not exist back then. Diffie would work with Martin Hellman and Ralph Merkle in search of such a system.

Today, both the RSA (Rivest-Shamir-Adleman) and the ECDSA (Elliptic Curve Digital Signature Algorithm) are two popularly used algorithms for public key cryptography.

The security of the RSA algorithm relies on the practical difficulty of factoring the product of two large prime numbers - "the factoring problem". An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret, messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers. This algorithm is relatively slow, but there are no published methods to decypher this system if a large enough key is used.

The ECDSA uses elliptic curves. It can provide the same level of security as other public key algorithms with smaller key sizes, which is the reason of its popularity. This is the Digital Signing Algorithm used by Bitcoin, specifically the secp256k1 curve.

Consensus mechanisms

Blockchain networks are essentially distributed and decentralized databases consisting of many nodes (computers). In a decentralized environment, common issues are:

- How do all nodes agree on what the current and future state of user account balances and contract interactions is?
- Who gets to add new blocks/transactions to a chain? How do we know any blocks added are "valid"?
- How is this system coordinated without any official coordinator?

All of this is done thanks to consensus mechanisms.

The blockchain consensus mechanism typically means that at least 51% of nodes are in agreement over the current global state of network (Ahmad et al, 2021). Essentially, these are rules that distributed, decentralized blockchain follows in order to stay in agreement over what is considered valid. There are many consensus mechanisms, but the two most famous ones are: proof-of-work (PoW) and proof-of-stake (PoS).

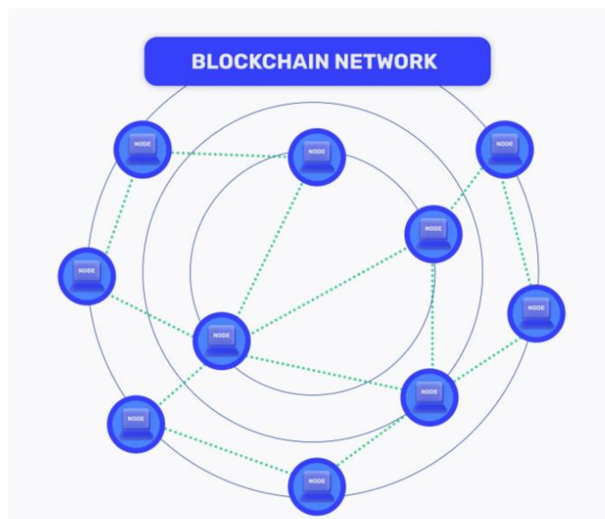


Figure 1 – Abstract scheme of the blockchain network

Proof of work & mining

Proof-of-work is the consensus mechanism that allows decentralized networks like Bitcoin and (previously) Ethereum to come to consensus, or agree on things like account balances and the order of transactions (Ali et al, 2023). This prevens "double spending" and ensures the followin of the blockchain rules, making the PoW network resilient to malicious attacks.

The main consensus rules for the PoW are the following:

- There must not be double spending, and
- The "longest" chain will be the one the rest of the nodes will accept as the one "true" chain - Nakamoto Consensus.

The consensus mechanism ends up being the security mechanism the network needs, because it ensures that every node on it is following the consensus rules. In PoW, mining represents the work.

Mining is the process of creating a block of transactions to be added to blockchain.

In proof-of-work consensus, nodes in the network continuously attempt to extend the chain with new blocks - these are the miners, nodes that contain mining software (Antonopoulos & Wood, 2018). Miners are in charge of extending a blockchain by adding blocks that contain "valid" transactions. In order to add a block, the network will ask miners for their "proof-of-work".

A proof-of-work-based system will typically require miners produce an output in a very difficult-to-get target range. Valid proof-of-work once looked like this in the Bitcoin network:

0000000000000000000043f43161dc56a08ffd0727df1516c987f7b187f5194c6

Figure 2 – The look of once valid proof-of-work in the Bitcoin network

To get an output like this, automated mining software does the following: takes a piece of data (i.e. the previous block header + new transactions to add to a chain) and hashes it. If the hash output is bellow a target difficulty, then the miner has found the answer to the puzzle: a valid proof of work.

The proof-of-work shown above has 19 leading zeroes, and since the range of each possible character per space is in hexadecimal, this means that there are 1/16 character possibilities per space.

The hash outputs for SHA-256 are in hexadecimal, which means there are 1/16 possible characters per space - a-f in letters and 0-9 in decimals = 16 total possibilities. This means that finding one 32-byte SHA-256 output that has just one leading zero will take on average 16 tries (Banerjee et al, 2018).

Finding an output with 2 leading zeros increases the average number of attempts to 256 - 16 possible characters in the first spot * 16 possible characters in the second spot. Finding 19 leading zeros will take, on average, 16^{19} attempts, which equals to 7555786372591432341913600000000000000000000 attempts.

Proof-of-work networks will typically have some sort of target_difficulty. In order for a miner to add a new block, they must find a proof-of-work lower than the network target difficulty. Finding such a difficult-to-find output is proof enough that a miner expended considerable resources to secure the network.

The proof-of-work mining algorithm looks like this:

- Take current block's block header, add mempool transactions
- Append a nonce, starting at nonce = 0
- Hash data from #1 and #2
- Check hash versus target difficulty (provided by protocol)
- If hash < target, puzzle is solved! Get rewarded.
- Else, restart process from step #2, but increment nonce

The miner nodes in a proof-of-work network will perform this algorithm regularly. This gives the network a way to recognize the true state and the validity of the proposed transactions following the consensus rules. As long as the majority of nodes on the network follow the consensus rules, the blockchain remains secure and resistant to attacks, ensuring that only valid and verified transactions are added to the distributed ledger, thus maintaining its integrity and trustworthiness. In exchange for large amounts of energy and hardware upkeep required to run mining software, miners receive currency as a reward.

Blockchain structure

A blockchain is a distributed database of a list of validated blocks (Bashir, 2018). Each block contains data in the form of transactions and each block is cryptographically tied to its predecessor, producing a "chain". Each blockchain consists of nodes.

A blockchain has nodes scattered all over the world all acting together in real-time. There is no central administrator, say a "supernode", responsible for verifying any changes to the state of data, all nodes are equal members of the network. This means that the network will perform the same, no matter what node is interacted with to update data. In other words, blockchains are peer-to-peer networks.

A valid hash for a blockchain is a hash that meets certain requirements. The number of leading zeros required is the difficulty. The process of finding valid hash outputs, via changing the nonce value, is called mining. A miner starts a "candidate block" with a nonce of 0 and keeps incrementing it by 1 until it finds a valid hash.

Since data is an input variable for the hash of each block, changing the data will change that block's hash. Blockchains like Bitcoin and Ethereum, protect the integrity of any data held inside blocks in their chains: manipulating data in a block that has been nested deeply in the chain is almost impossible.

In Bitcoin, Merkle trees are used to store every transaction mined on the Bitcoin network. Merkle tree is a data structure that represents a collection of hash values reduced to a single hash.

Each letter represents a hash. The combined letters represent concatenated hashes that have been combined and hashed to form a new hash.

Over a series of steps, the eight leaf hashes A, B, C, D, E, F, G, and H are combined to create a single, unique hash that allows efficient checking for inconsistencies without having to look at each individual data point.

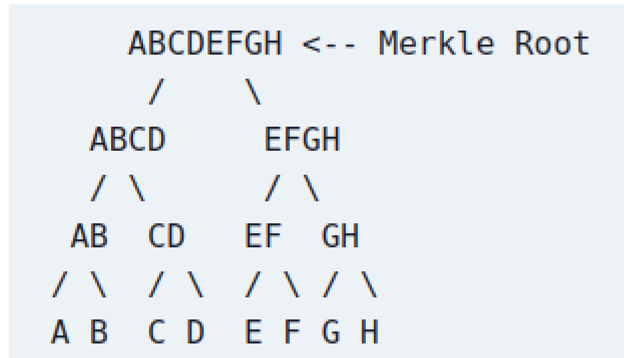


Figure 3 – Visual representation of Merkle tree

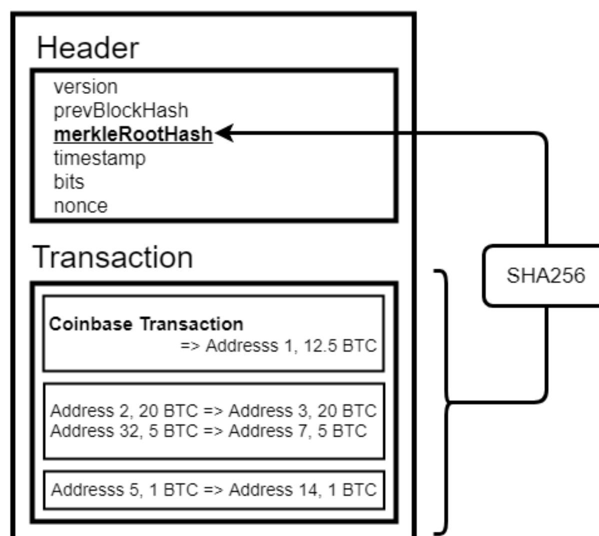


Figure 4 – The architecture of a Bitcoin block

The figure above shows the architecture of the Bitcoin block. All of the transactions per block are arranged into a big Merkle tree. Merkle tree's root hash actually gets commited into the block.

By committing the root hash of the tree, the transaction data can be stored off-chain (full nodes, for example, store these transaction records on a LevelDB integrated into all full nodes).

A main design purpose behind using Merkle trees to commit a lot of data elements (typically transactions) per block is to keep the size of the

blockchain as small as possible. Given the nature of their usage, blockchains grow perpetually. Keeping the blockchain size from becoming bloated means more people can support running full nodes which helps network decentralization (Dabbagh et al, 2019).

UTXO & account models

With traditional web2 server based platforms, keeping track of user data and information is actually a lot easier than it is on the blockchain. This is because there is a single centralized server that stores the state of user accounts. There is no need for consensus or resolving discrepancies since there is only one central place that stores information.

However, when moving to a decentralized system, the problem of storing user balances becomes complicated. Decentralized networks like Bitcoin and Ethereum need specific models for keeping track of the state of users. Bitcoin uses the UTXO (Unspent Transaction Output) model to keep track of user balances. Ethereum and other EVM chains use the account model to keep track of user balances.

The account model tracks the balances of users based on their overall account state, without knowing what constitutes the actual balance itself. This model is a lot like a classical bank account model.

In the account model, the ownership of cryptocurrency is determined by the account's private key, which corresponds to a unique public key or address. When a user initiates a transaction, they sign it with their private key to prove ownership and authorize the movement of funds from their account. This model simplifies the transaction process and is more intuitive for developers building decentralized applications, as it resembles the familiar ledger system used in traditional banking. However, it also comes with challenges, such as the need for more complex protocols to prevent issues like double-spending. The choice between the UTXO and account models reflects different design philosophies and trade-offs in the realm of blockchain architecture (Buterin, 2013).

The Unspent Transaction Output (UTXO) model is a fundamental concept underpinning the functioning of the Bitcoin blockchain. In the Bitcoin network, transactions are represented as a chain of inputs and outputs. Each output of a transaction is a certain amount of bitcoin, and these outputs serve as the inputs for future transactions. The UTXO model is designed to keep track of the ownership of Bitcoin and prevent double-spending. In simple terms, a UTXO is essentially an unspent output of a transaction that can be used as an input for a new transaction. This model contrasts with the account-based model used by traditional banking

systems, where an account balance is maintained, and transactions involve debiting and crediting these balances.

In the UTXO model, the ownership of Bitcoin is determined by the ability to provide a valid digital signature corresponding to the public key associated with a UTXO. When a user initiates a transaction, they must reference one or more UTXOs as inputs, providing the required digital signatures to prove ownership. The outputs of this transaction become new UTXOs, which can be spent in future transactions. This model adds a layer of security to the Bitcoin protocol by ensuring that every transaction input is indeed an unspent and valid output from a previous transaction. It also contributes to the decentralized and trustless nature of the Bitcoin network, as the entire transaction history is publicly accessible and verifiable by anyone on the blockchain (Buterin, 2013).

Ethereum

Bitcoin was the first blockchain-based decentralized network ever. It popularized the use of Merkle trees for scalable transaction inclusion. Ethereum also uses Merkle trees but since Ethereum is a completely different design, it also uses one other important tree data structure for some of its data storage needs: Patricia Merkle Tries. Unlike Bitcoin, Ethereum uses the Keccak256 hash function.

Ethereum is a deterministic but practically unbounded state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to that state (Dange & Nitnaware, 2023).

Essentially, Ethereum can be seen just like any other computer in the world. This computer has some major features that make it unique:

- It is the first global singleton machine ever, that fundamentally is not localized (not located on any physical machine in the world). Ethereum does not reside in any single machine, with no physical presence anywhere.
- Ethereum is totally censorship resistant. No authority, government, corporation or a group of individuals is behind the Ethereum computer. No one owns it, can shut it off or can us it as a privileged user.
- Ethereum is ubiquitous and accessible anywhere there is Internet connection.
- Natively multi-user, with a practically infinite range possible for account creation - 2^{160} accounts.

Since Ethereum keeps track of a larger amount of state data than Bitcoin, its block architecture is completely different.

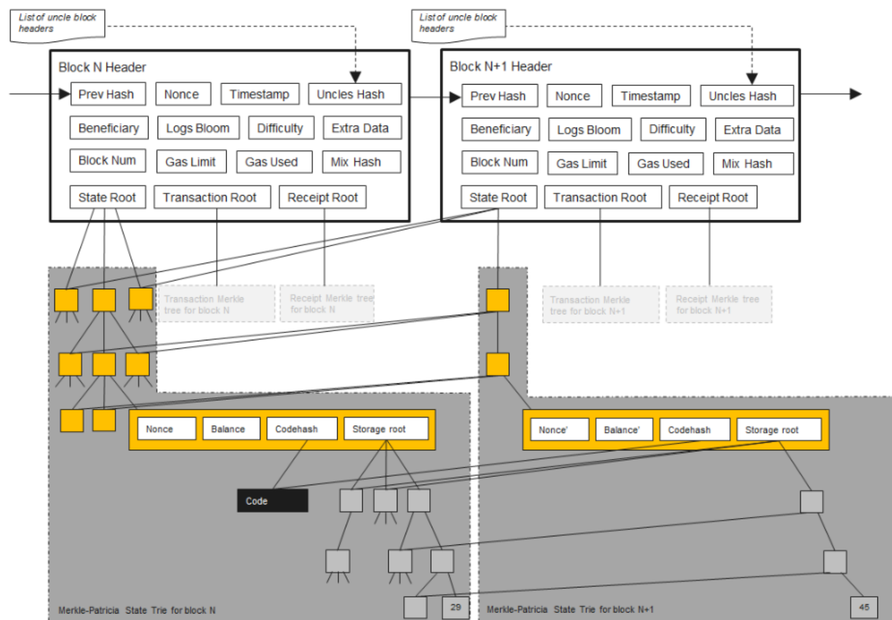


Figure 5 – The architecture of the Ethereum block

Ethereum makes use of a data structure called radix trie (Patricia trie, radix tree) and combines it with the Merkle tree structure to create a Patricia Merkle Trie.

Trie comes from the word "retrieval", meaning that radix trie is a tree-like data structure that is used to retrieve a string value by traversing down a branch of nodes that store associated references (keys) that together lead to the end value that can be returned.

A Patricia Merkle trie (PMT) is a data structure that stores key-value pairs, just like a hash table. In addition to that, it is also used verify data integrity and the inclusion of a key-value pair. PMTs groups similar-value nodes together in the tree. That way, searching for "HELP" leads you along the same path as searching for "HELLO" - the first three letters are shared entries of different words. It is very good for space efficiency and read/write efficiency. Patricia is an acronym: P - Practical; A - Algorithm; T - To; R - Retrieve; I - Information; C - Coded; I - In; and A - Alphanumeric.

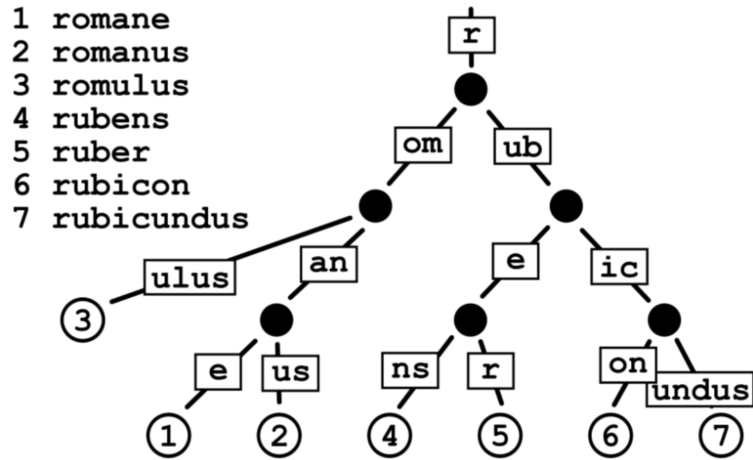


Figure 6 – An example of the radix trie data structure

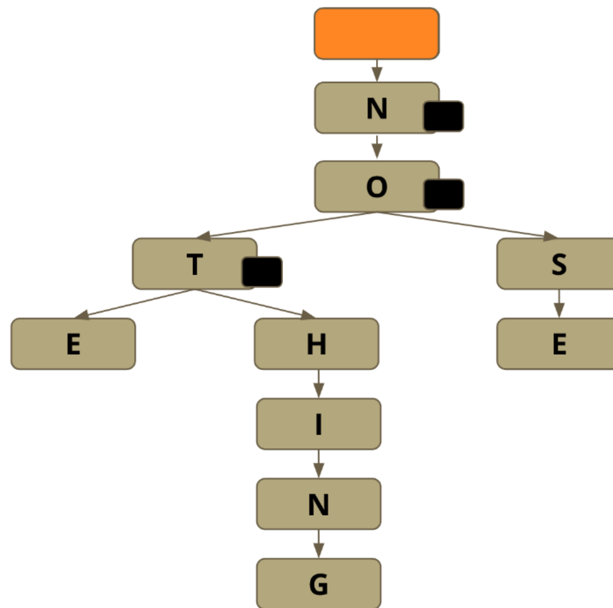


Figure 7 – An example of the Patricia Merkle trie data structure

Ethereum stores two types of data: permanent and ephemeral. It makes sense that permanent data, like mined transactions, and ephemeral data, like Ethereum accounts (balance, nonce, etc), should be stored separately. Merkle trees are perfect for permanent data. PMTs are perfect for ephemeral data, which Ethereum is in plenty supply of.

Unlike transaction history, the Ethereum account state needs to be frequently updated. The balance and nonce of accounts is often changed, and what is more, new accounts are frequently inserted, and keys in storage are frequently inserted and deleted.

The Ethereum block header contains many pieces of data.The block header is the hash result of all of the data elements contained in a block. It is like a gift-wrap of all the block data.

Looking at the Ethereum architecture diagram at the beginning of this chapter, the block header ends up hashing all of the data properties of the block. It also includes:

- State Root: the root hash of the state trie,
- Transactions Root: the root hash of the block's transactions, and
- Receipts Root: the root hash of the receipts trie.

The state trie acts as a mapping between addresses and accounts states. It can be seen as a global state that is constantly updated by transaction executions. All the information about accounts is stored in the world state trie and information can be retrieved by querying it.

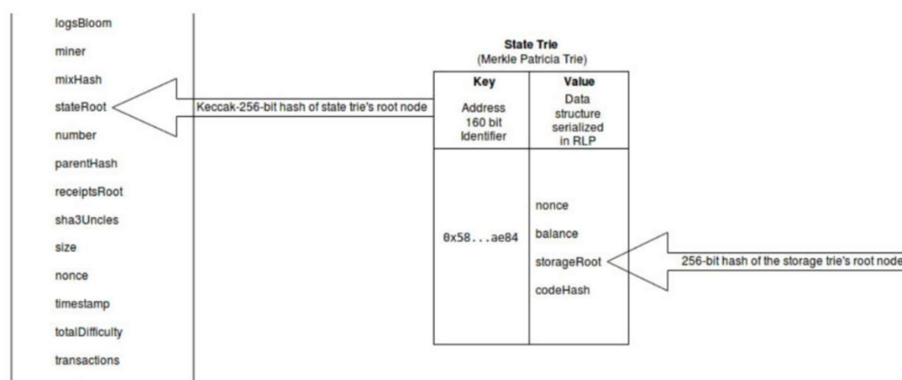


Figure 8 – An example of the state trie and its integration with the Ethereum block

The transaction trie records transactions in Ethereum. Once the block is mined, the transaction trie is never updated. Each transaction in

Ethereum records multiple pieces specific to each transaction such as gasPrice and value.

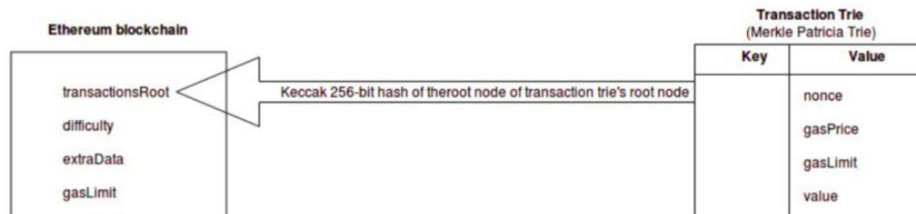


Figure 9 – An example of the transaction trie and its integration with the Ethereum block

The transaction receipt trie records receipts (outcomes) of transactions. It contains data including gasUsed, logs and events emitted. Once the block is mined, the transaction receipt trie is never updated.

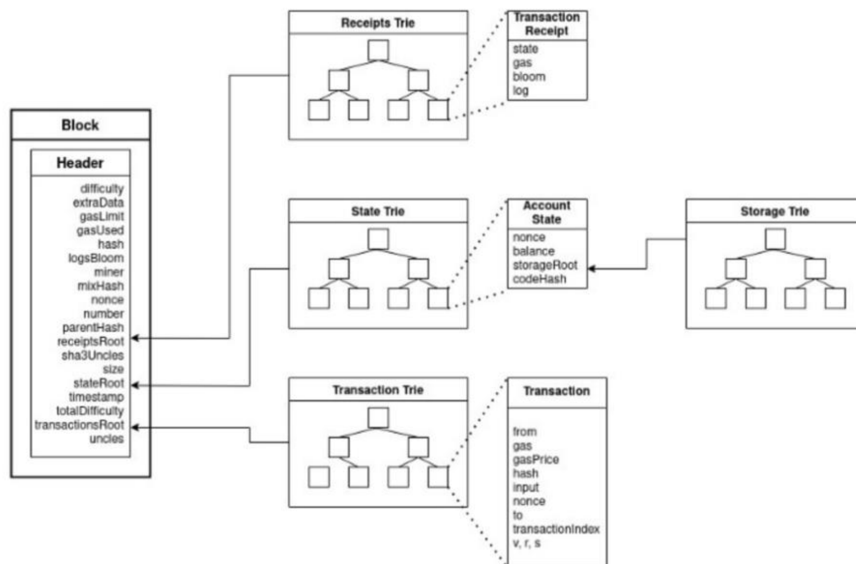


Figure 10 – Visualizaton of how the tries end up being commited in every block via their root hash

Proof of stake

Ethereum transitioned to PoS on September 15th, 2022. This transition is known as "The Merge". This was a massive migration that was always in the roadmap and original planning for Ethereum, but required coordination from the entire network to execute.

Proof-of-stake is a totally different mechanism than proof-of-work that enables Ethereum to be more secure, less energy intensive and more scalable.

In order to become a miner in PoW, there are large energy requirements, which makes it difficult for any individual to compete with the existing mining warehouses that are dedicating millions of dollars of resources to mining. However, in proof-of-stake, the energy requirement to become a validator is much lower and can be done by individuals without a high overhead energy cost. This encourages more users to become validators, decreasing the centralization risk, and thereby increasing the security of the network.

Instead of using mass amounts of electricity, validators are required to stake 32ETH by depositing it into a contract to have the ability to validate blocks. This staked ETH is used as collateral against bad actors in the network. If any given validator acts dishonestly or maliciously, they put themselves at risk of losing their staked ETH.

Rather than all validators competing at the same time for the next block, the network randomly selects a validator to propose a block every 12 seconds, all the other validators verify that the proposed block is correct, and the cycle repeats (Stanišić, 2023).

One of the largest ways that PoS affects Ethereum developers is with a new framework for block finality. Finality in blocks refers to how confident you are that the given block will not change or get forked away. For blocks that have been on the network for a very long time (older blocks), it is extremely unlikely that it will be removed from the canonical chain and therefor has high finality.

Proof of stake introduced 2 new levels of finality that developers should consider when requesting data from the network: safe and finalized. Here is an overview of all “block tags”:

- earliest: The lowest numbered block the client has available. Intuitively, you can think of this as the first block created.
- finalized: The most recent crypto-economically secure block, that has been accepted by >2/3 of validators. Typically finalized in two epochs (64 blocks). Cannot be reorganized outside manual intervention driven by community coordination. Intuitively, this block is very unlikely to be reorganized.
- safe: The most recent crypto-economically secure block, typically safe in one epoch (32 blocks). Cannot be re-orged outside manual intervention driven by community coordination. Intuitively, this block is unlikely to be re-orged.

- latest: The most recent block in the canonical chain observed by the client, this block may be re-orged out of the canonical chain even under healthy/normal conditions. Intuitively, this block is the most recent block observed by the client.
- pending: A sample next block built by the client on top of latest and containing the set of transactions usually taken from local mempool. Intuitively, you can think of these as blocks that have not been mined yet.

Gas on Ethereum

The cost of operations on Ethereum are fixed and measured in a unit called "gas". The price of gas is what constantly changes. This means that the energy requirements to mine any given block are significantly lower than that of PoW.

In August 2021, there was an Ethereum Improvement Proposal (EIP) to improve the calculation of gas prices on Ethereum, known as EIP-1559. Historically, gas prices on Ethereum have been unpredictable and at times astronomically high, making transactions inaccessible to most people. EIP-1559 changed the mechanism for setting the gas price, making participating in Ethereum blockchain accessible to pretty much everyone.

Just like every currency in the world, Ethereum also has different denominations thar are used to express smaller values. 1 ether is equal to 10^{18} wei (the smallest denomination of ether) or 10^9 gwei.

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

Figure 11 – Table with relevant denominations for ether

Every block has a maximum amount of gas that can be used within it. This is how a number of transactions included within a block are determined. Every block has the capacity to use 30 million gas but has a

target of 15 million gas total. The price of gas is determined by the amount of demand for transactions (or block space), where demand is measured by how filled the previous block was relative to the target gas.

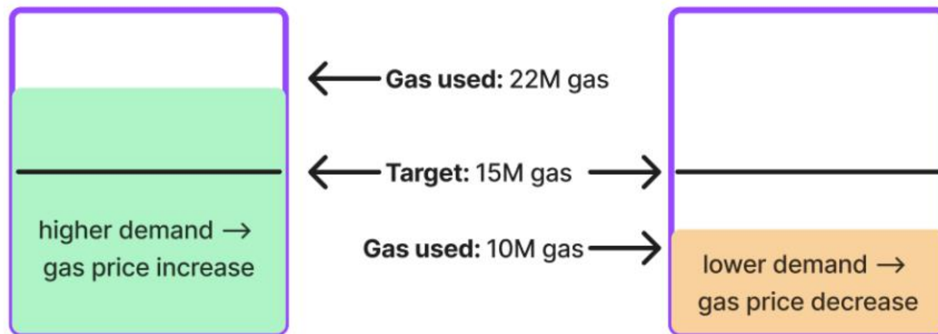


Figure 11 – Example of two different blocks with different demand quantity

The network first sets a base fee; in an ideal world, this base fee would result in 15 million gas getting used in a block, no more, no less. However, what happens in practice is the actual gas can be above or below the target gas.

When blocks are above the target, the gas price (or base fee) is automatically increased, increasing the cost and barrier to entry for sending transactions and thereby reducing the number of people who are competing to fill the block. When the block is below the target, the base fee is lowered to incentivize people to transact by lowering the barrier to entry for paying for a transaction.

This base fee helps users select an efficient gas amount that is likely to get their transaction mined rather than wasting tons of money on unnecessarily high gas prices like in the past. These mechanisms also make it easy to predict future gas prices by looking at how "full" the previous blocks were.

Instead of going straight into the miners pocket, the base fee actually gets burned. There are several reasons why the base fee is burned instead of being given to the miner:

- This prevents the miner from circumventing the payment of the base fee since they have to pay at least base fee times the number of transactions for the block that the mine, and
- Burning ether also creates a deflationary pressure on ether as an asset since supply is taken out of the market.

Since the base fee is entirely burned, the new incentive for miners is now known as the miner tip. In a perfect world, the miner tip is the minimum amount that the miner is willing to accept in order to execute the transaction. This tip was originally set as 1gwei but can fluctuate depending on how full blocks are. Since the target gas value in blocks is 15M, in general, so long as blocks are hitting or near the target amount, there will always be room to add more transactions within a block. This is why the miner tip does not need to be insanely high to get some transaction included.

Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. It is a language that closely resembles other popular programming languages like C++, Python and JavaScript. Solidity is statically-typed (variables must be defined at compile time) and supports inheritance, libraries and complex user-defined types. It is a programming language used to write smart contracts.

A smart contract is a set of promises, specified in a digital form, including protocols within which the parties perform on these promises (Szabo, 1996). Basically, smart contracts are typical contracts, but in a digital form, and they have stronger enforcement parameters (Szabo, 1997).

A smart contract is simply a program that runs on the Ethereum computer. More specifically, a smart contract is a collection of code (functions) and data (state) that resides on a specific address on the Ethereum blockchain. These are written in Solidity which means they must be compiled into bytecode first in order to be Ethereum compatible.

Smart contracts are permisionless (anyone can deploy them to Ethereum) and composable (they are gloably available via Ethereum).

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.4;
3
4  address public owner;
5  bool public isHappy;
6  uint public x = 10;
7  int public y = -50;
8
9  contract MyContract {
10     constructor(address _owner, bool _isHappy) {
11         owner = _owner;
12         isHappy = _isHappy;
13     }
14 }
```

Figure 12 – A sample of a Solidity smart contract

The features of Solidity will be explained on the example listed as Figure 12.

Line 1: Specifies what type of license will be used and determines what license rules fall on that specific smart contract.

Line 2: The word **pragma** defines the version of Solidity that will be used for writing the smart contract. Solidity uses semantic versioning.

Lines 4-7: Define state variables that will be used throughout the writing of the smart contract. Variables in Solidity can have private, public and internal visibility. Numbers in Solidity can be **int** (interger) and **uint** (unsigned integer).

Lines 9-14: The scope of the contract. The contract keyword behaves very simliar to the class keyword of JavaScript.

Lines 10-13: The **constructor()** function is called only once during deployment and completely discarded thereafter. It is used to specify the state when deploying a contract.

There are many data types in Solidity: boolean (**bool**), **string**, integers (**uint** and **int**), **bytes**, **enums**, **arrays**, **mappings**, and **structs**.

A solidity-specific type of variable is called **address**. There are two types of this variable: **address** and **address payable**. These two types are more than just some string holding Ethereum address value, they are first-class types, meaning that they have a number of methods and function that can be called upon them.

Integration with Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is a runtime environment that executes smart contracts on the Ethereum blockchain. The Ethereum Virtual Machine is a crucial component of the Ethereum network, enabling the execution of decentralized applications (DApps) by processing and validating smart contracts code. It plays a central role in ensuring the decentralized and trustless nature of the Ethereum platform by allowing participants to execute code without the need for a central authority. Smart contracts written in languages like Solidity are compiled into bytecode that can be executed by the Ethereum Virtual Machine.

After a contract has been compiled, the bytecode of that contract is sent to the EVM. For a contract containing a simple while loop that increments a variable of type integer five times, the bytecode looks like this:

```
6080604052348015600f57600080fd5b5060a58061001e6000396000f3fe6080604052348015600f57600080fd5b50
600436106028576000356 0e01c8063a92100cb14602d575b600080fd5b60336049565b60405180828152602001915 0
5060405180910390f35b600080600090506000809050 5b60058210156067578181019050 6056565b809250505090 56
fea2646970667358221220 58d7e11ff1d36fc53779562e305af3c9180b2ab8dccfe6d234fa50420908a5d864736f6c
63430006030033
```

Figure 13 – The bytecode of the while loop

The bytecode contains opcodes and operands. This bytecode looks like this after looking up the EVM operation codes:

```
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH1 0xF JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST POP PUSH1 0xA5 DUP1 PUSH2 0x1E PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN INVALID PUSH1 0x80
PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH1 0xF JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP
PUSH1 0x4 CALLDATASIZE LT PUSH1 0x28 JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4
0xA92100CB EQ PUSH1 0x2D JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH1 0x33 PUSH1 0x49
JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40
MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1 0x0 DUP1 PUSH1 0x0 SWAP1 POP PUSH1 0x0 DUP1
SWAP1 POP JUMPDEST PUSH1 0x5 DUP3 LT ISZERO PUSH1 0x67 JUMPI DUP2 DUP2 ADD SWAP1 POP PUSH1
0x56 JUMP JUMPDEST DUP1 SWAP3 POP POP POP SWAP1 JUMP INVALID LOG2 PUSH5 0x6970667358 0x22 SLT
KECCAK256 PC 0xD7 0xE1 0x1F CALL 0xD3 PUSH16 0xC53779562E305AF3C9180B2AB8DCCFE6 0xD2 CALLVALUE
STATICCALL POP TIMESTAMP MULMOD ADDMOD 0xA5 0xD8 PUSH5 0x736F6C6343 STOP MOD SUB STOP CALLER
```

Figure 14 – The look of the bytecode after transposing the values of opcodes and operands

Conclusion

In the ever-evolving landscape of blockchain technology, the significance of the Solidity programming language is underscored by its integral role in platforms like Bitcoin and Ethereum. Bitcoin, the pioneering cryptocurrency, employs a blockchain data structure to create a secure, decentralized ledger of transactions. Solidity, however, takes the concept of blockchain a step further within the Ethereum ecosystem, enabling the development of smart contracts. These self-executing contracts, written in Solidity, automate and enforce predefined rules, facilitating a wide array of decentralized applications. Ethereum's versatility, driven by Solidity, extends the capabilities of blockchain beyond a mere medium of exchange, transforming it into a decentralized computing platform with applications spanning finance, gaming, and decentralized finance (DeFi).

As industries recognize the potential benefits of blockchain, its integration is becoming increasingly prevalent across sectors. From enhancing the traceability of goods in supply chain management to revolutionizing traditional financial systems through decentralized finance applications, the transformative impact of blockchain is gaining

momentum. Solidity's role in facilitating the creation and execution of smart contracts plays a crucial part in this evolution, offering developers a powerful tool to build decentralized applications that foster trust, transparency, and efficiency. The ongoing convergence of Solidity, blockchain data structures, and real-world applications suggests a promising future where decentralized technologies redefine how industries operate and interact.

References

Abou Jaoude, J. & Saade, R.G. 2019. Blockchain Applications – Usage in Different Domains. *IEEE Access*, 7, pp.45360-45381. Available at: https://doi.org/10.1109/ACCESS.2019.2902501.

Ahmad, D., Lutfiani, N., Rizki Ahmad, A.D.A., Rahardja, U. & Aini, Q. 2021. Blockchain Technology Immutability Framework Design in E-Government. *Jurnal Administrasi Publik (Public Administration Journal),* 11(1), pp.32-41. Available at: https://doi.org/10.31289/jap.v11i1.4310.

Ali, I.M., Lasla, N., Abdallah, M.M., Erbad, A. 2023. SRP: An Efficient Runtime Protection Framework for Blockchain-based Smart Contracts. *Journal of Network and Computer Applications*, 216, art.number:103658. Available at: https://doi.org/10.2139/ssrn.4050282.

Antonopoulos, A. & Wood, G. 2018*. Mastering Ethereum: Building Smart Contracts and DApps, 1st Edition*. Newton, MA, USA: O'Reilly Media. ISBN: 978-1491971949.

Banerjee, M., Lee, J. & Raymond Choo, K.-K. 2018. A blockchain future for internet of things security: a position paper. *Digital Communications and Networks*, 4(3), pp.149-160. Available at: https://doi.org/10.1016/j.dcan.2017.10.006.

Bashir, I. 2018. *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained, 2nd Edition*. Birmingham-Mumbai: Packt Publishing; 2nd Revised edition. ISBN: 978-1788839044.

Buterin, V. 2013. A next generation smart contract & decentralized application platform. *Whitepaper.io* [online]. Available at: https://whitepaper.io/document/5/ethereum-whitepaper [Accessed: 27 September 2023].

Dabbagh, M., Sookhak, M. & Safa, N.S. 2019. The Evolution of Blockchain: A Bibliometric Study. *IEEE Access*, 7, pp.19212-19221. Available at: https://doi.org/10.1109/ACCESS.2019.2895646.

Dange, S. & Nitnaware, P. 2023. Secure Share: Optimal Blokchain Integration in IoT Systems. *Journal of Computer Information Systems*, April 12. Available at: https://doi.org/10.1080/08874417.2023.2193943.

Nakamoto, S. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. *ResearchGate* [online]. Available at: https://www.researchgate.net/publication/228640975_Bitcoin_A_Peer-to-Peer_Electronic_Cash_System [Accessed: 27 September 2023].

Stanišić, S. 2023. *Primena Solidity programskog jezika u Blockchain tehnologiji.* BS thesis. Belgrade, Serbia: University of Defence (in Serbian).

Szabo, N. 1996. Smart Contracts: Building Blocks for Digital Markets. *Phonetic Sciences, Amsterdam* [online]. Available at: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html [Accessed: 27 September 2023].

Szabo, N. 1997. The Idea of Smart Contracts. *Phonetic Sciences, Amsterdam* [online]. Available at: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html [Accessed: 27 September 2023].

La utilización del lenguaje de programación Solidity en cadena de bloques (blockchain)

Sava S. Stanišić[a], *Hristina* N. Stojanović[b], *Igor* Lj. Đorđević[c]

[a] Fuerzas Armadas de Serbia, Fuerza Aérea y Defensa Antiaérea, 98.a Brigada de las Fuerzas Aéreas, Lađevci, República de Serbia, **autor de correspondencia**

[b] Fuerzas Armadas de Serbia, Fuerza Aérea y Defensa Aérea, 126.a Brigada ASEWG, Belgrado, República de Serbia

[c] Fuerzas Armadas de Serbia, Estado Mayor Conjunto, Departamento de Telecomunicaciones e Informática, Belgrado, República de Serbia; Universidad Megatrend, Facultad de Informática, Belgrado, República de Serbia

CAMPO: ciencias de computación, IT
TIPO DE ARTÍCULO: artículo de revisión

Resumen:

Introducción/objetivo: Este trabajo proporciona una descripción general completa de la tecnología blockchain, aclarando sus principios fundamentales y cómo garantiza la transparencia, la inmutabilidad y la descentralización. La integración de Solidity con blockchain se explora a través de un enfoque teórico.

Métodos: Este trabajo analiza meticulosamente los principios de blockchain, aclarando la transparencia, la inmutabilidad y la descentralización, mientras explora la integración de Solidity en un marco teórico, asegurando una comprensión integral de su intrincada relación y contribuyendo a una comprensión más amplia de la tecnología moderna de distribuición de registros.

Resultados: El producto resultante de este artículo será la obtención de conocimientos útiles sobre la tecnología que prácticamente da forma al mundo.

Conclusión: En conclusión, la adopción de Solidity como lenguaje de programación en la tecnología blockchain ha demostrado ser fundamental, ya que mejora la funcionalidad de los contratos inteligentes y la seguridad general del sistema. Sus características especializadas la convierten en una herramienta indispensable para los desarrolladores que navegan por las complejidades de las aplicaciones descentralizadas.

Palabras claves: blockchain, Bitcoin, Ethereum, Solidez, descentralización.

Применение языка программирования Solidity в технологии blockchain

Сава С. Станишич[а], *Христина* Н. Стоянович[б], *Игорь* Л. Джорджевич[в]

[а] Вооруженные силы Республики Сербия, Военная авиация и противовоздушная оборона, 98-ая авиационная бригада, Ладжевци, Республика Сербия, **корреспондент**

[б] Вооруженные силы Республики Сербия, Военная авиация и противовоздушная оборона, 126-ая авиационная бригада ВНОС, г. Белград, Республика Сербия

[в] Вооруженные силы Республики Сербия, Генштаб, Управление информатики и телекоммуникаций (J-6), г. Белград, Республика Сербия; Университете „Мегатренд“, факультет вычеслитетельных наук, г. Белград, Республика Сербия

РУБРИКА ГРНТИ: 20.15.05 Информационные службы, сети, системы в целом,
81.93.29 Информационная безопасность. Защита информации
ВИД СТАТЬИ: обзорная статья

Резюме:

Введение/цель: В данной статье представлен всесторонний обзор блокчейн технологии, разъясняются ее основополагающие принципы и то, как она обеспечивает прозрачность, неизменность и децентрализацию. Интеграция Solidity с блокчейном исследуется с помощью теоретического подхода.

Методы: В данной статье представлены принципы blockchain технологии. Теоретический подход и фрагменты кода на практике показывают, как Solidity сочетается с этой технологией и почему она является фундаментом развития современных технологий и многих отраслей промышленности.

Результаты: В результате исследования получены полезные сведения о технологии, которая встречается практически во всех сферах современного мира.

Выводы: Внедрение Solidity в качестве языка программирования в блокчейн технологию оказалось ключевым фактором в повышении функциональности смарт-контрактов и общей безопасности системы. Его специальные характеристики делают его незаменимым инструментом для разработчиков, занимающихся сложностями децентрализованных приложений.

Ключевые слова: блокчейн, биткоин, Ethereum, Solidity, децентрализация.

Примена програмског језика Solidity у *blockchain* технологији

Сава С. Станишић[а], *Христина* Н. Стојановић[б], *Игор* Љ. Ђорђевић[в]

[а] Војска Србије, Ратно ваздухопловство и противваздухопловна одбрана, 98. ваздухопловна бригада, Лађевци, Република Србија, **аутор за преписку**

[б] Војска Србије, Ратно ваздухопловство и противваздухопловна одбрана, 126. бригада ВОЈИН, Београд, Република Србија

[в] Војска Србије, Генералштаб, Управа за телекомуникације и информатику (J-6), Београд, Република Србија; Мегатренд Универзитет, Факултет за компјутерске науке, Београд, Република Србија

ОБЛАСТ: рачунарске науке, ИТ
КАТЕГОРИЈА (ТИП) ЧЛАНКА: прегледни рад

Сажетак:

Увод: У раду је представљена blockchain технологија, њени основни принципи и начин на који се осигурава транспарентност, непроменљивост и децентрализација. Интеграција програмског језика Solidity са blockchain технологијом објашњена је теоријским приступом.

Методе: Расветљени су принципи blockchain технологије. Теоријским приступом и исечцима кода показано је како се Solidity интегрише са овом технологијом и зашто представља стуб развоја савремених технологија и многобројних индустријских грана.

Резултати: Добијене су корисне информације о технологији која је примењена у свим областима данашњег света.

Закључак: Усвајање програмског језика Solidity у blockchain технологији показало се кључним, јер побољшава функционалност паметних уговора и укупну сигурност система. Његове специјализоване карактеристике чине га неопходним алатом за програмере који се крећу кроз комплексност децентрализованих апликација.

Кључне речи: blockchain, Bitcoin, Ethereum, Solidity, децентрализација.

Paper received on: 27.09.2023.
Manuscript corrections submitted on: 03.03.2024.
Paper accepted for publishing on: 04.03.2024.

© 2024 The Authors. Published by Vojnotehnički glasnik / Military Technical Courier (www.vtg.mod.gov.rs, втг.мо.упр.срб). This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/rs/).