

SMART CONTRACT DEVELOPMENT METHODS AND ANALYSIS OF UNDERLYING ALGORITHMS

Razzakov Sherbek Tog'aymurod ugli
Normurodov Chori Begaliyevich.
Termez State University

Abstract:

Smart contracts are self-executing agreements deployed on blockchain platforms. Their correct development and the choice of underlying algorithms play a crucial role in ensuring trust, efficiency, and security in decentralized systems. This thesis explores different methods for creating smart contracts and analyzes the algorithms used in various stages, such as consensus mechanisms, cryptographic functions, and execution models. Comparative analysis is performed on the basis of performance, scalability, and vulnerability resistance. The study provides insights for developers and researchers working on secure and efficient smart contract systems.

Keywords

Encryption, symmetric algorithm, data security, cryptanalysis, evaluation criteria, framework,

1. Introduction

With the emergence of blockchain technology, smart contracts have become a foundational element of decentralized applications (dApps). A smart contract is a program stored on a blockchain that automatically executes when certain conditions are met. Ethereum popularized this concept, but many other platforms like Solana, Polkadot, and Binance Smart Chain now support smart contract functionality.

The development of smart contracts requires not only proper programming but also an understanding of the algorithms and mechanisms behind their execution. This thesis focuses on:

- Common development methods for smart contracts.
- Key algorithms used in smart contract operation.
- Security and performance analysis of these algorithms.

2. Methods of Smart Contract Development

There are various approaches to building smart contracts, depending on the blockchain platform and desired functionality. The main methods include:

2.1. Using High-Level Languages

- **Solidity** (for Ethereum-based chains)

- **Rust** (for Solana)
- **Vyper** (for Ethereum, with a focus on security)

2.2. Visual Programming Tools

- Tools like **Remix IDE**, **Chainlink Starter Kit**, and **OpenZeppelin Wizard** enable low-code or no-code creation of smart contracts.

2.3. Framework-Based Development

- Using development frameworks like:
 - **Truffle**
 - **Hardhat**
 - **Brownie**

These frameworks provide testing, deployment, and debugging environment

3. Algorithms Used in Smart Contracts

Smart contracts rely on several algorithmic components, including:

3.1. Consensus Algorithms

These algorithms ensure agreement on the state of the blockchain:

- **Proof of Work (PoW)** – Used in Ethereum before the Merge.
- **Proof of Stake (PoS)** – Used in Ethereum 2.0 and others.
- **Delegated Proof of Stake (DPoS)** – Used in EOS, TRON.

3.2. Cryptographic Algorithms

These provide data integrity and identity verification:

- **SHA-256** – For hashing (Bitcoin).
- **Keccak-256** – For hashing (Ethereum).
- **ECDSA (Elliptic Curve Digital Signature Algorithm)** – For signing transactions.

3.3. Execution and Gas Algorithms

- **EVM (Ethereum Virtual Machine)** execution model.
- **WASM (WebAssembly)** – Used in Polkadot and EOS for contract execution.
- **Gas Calculation Algorithm** – Determines the cost of executing a contract.

4. Analysis and Comparison

Feature	Solidity (EVM)	Rust (Solana)	Vyper (Ethereum)
Language Complexity	Moderate	High	Low
Performance	Moderate	High	Low
Security Focus	Medium	Medium	High
Tooling Support	Strong	Growing	Limited
Execution Model	EVM	BPF/WASM	EVM

Algorithmic Analysis:

- **Consensus Algorithms:** PoS offers better scalability than PoW but can introduce new attack vectors (e.g., long-range attacks).
- **Cryptographic Algorithms:** ECDSA is widely used but quantum-vulnerable; future-proof alternatives like EdDSA or post-quantum algorithms are being considered.
- **Gas Algorithms:** Smart contracts must be optimized to reduce gas costs. Loops and heavy storage operations are particularly expensive.

5. Conclusion

The development of smart contracts requires an in-depth understanding of both programming paradigms and underlying algorithms. While tools and languages simplify the development process, security and efficiency depend heavily on algorithmic design. As blockchain platforms evolve, new languages and execution environments are emerging, requiring developers to adapt to changing technologies. The choice of algorithm, whether for consensus, cryptography, or execution, directly influences the performance, scalability, and trustworthiness of smart contract systems. Future research should focus on quantum-resistant cryptographic algorithms, gas optimization strategies, and formal verification tools to enhance the robustness of smart contracts.

References

1. Wood, G. (2014). *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Ethereum Project Yellow Paper.
2. Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*.
3. Buterin, V. (2015). *A Next-Generation Smart Contract and Decentralized Application Platform*.
4. Szabo, N. (1997). *The Idea of Smart Contracts*.
5. OpenZeppelin Documentation. (2023). Retrieved from <https://docs.openzeppelin.com>
6. Solidity Language Docs. (2023). Retrieved from <https://docs.soliditylang.org>
7. Solana Developer Docs. (2023). <https://docs.solana.com>