Implementation of Blockchain-based Systems

Author: Kamil Rafikov
Email: mailbox@kamil-rafik.com
WhatsApp: +996 755 439 777
Telegram: kamilrafikov

Updated at Novermber 8th, 2025.
Edition: 3 (final fact-checking not yet performed).

Table of Contents

Introduction

This document is written to demonstrate my theoretical knowledge of the topic of implementation of blockchain-based systems. It is not related anyhow to the projects performed by me at previous jobs. That's why you should not and cannot use the information provided further as for development of MVP as for development of production stage projects. However, I suppose it may be helpful for you when you will plan introduction of blockchain into your organizational and/or business processes.

When you really don't need blockchain?

You definitely don't need blockchain if data that you expect to place into it match all the following criteria: a) they should be accessed by very limited number of users, b) they may be or must be able to be changed completely and regularly, c) they need to be accessed in large amounts in real-time. Traditional database management systems allow to reach all these goals with much less issues.

Design approaches

Permissions management options: a) public permissionless; b) private permissioned (as in single organization as in consortium of organizations); c) hybrid (private chain processing, public chain anchoring).

Consensus management options: a) proof-of-work; b) proof-of-stake; c) delegated proof-of-stake; d) Byzantine Fault Tolerant consensus; e) proof-of-authority; f) hybrid.

Architecture options: a) monolithic; b) modular; c) secondary blockchain built on top of another one; d) separate blockchain built as a sidechain connecting via bridges to another one; e) sharded.

Cryptocurrency ownership management options: a) by transactions outputs; b) by account balances.

Private data management options: a) using zero-knowledge proofs to hide interactions between users (with optional keys for legal agencies to track interactions); b) keeping only hashes of data on chain, the data are stored and managed in external storage; c) keeping data on chain in encrypted form only, keys are stored and managed in external storage; d) using "chameleon" hash functions that allow authorized edits of chain data with keeping blocks valid; e) using protocol with consensus-level rules allowing authorized edits.

Complexity options of code executed on chain: a) basic scripts; b) smart contracts; c) specialized application-specific modules.

Oracles usage options: a) by direction of data flow; b) by centralization level; c) by data security level; d) by connectivity level between oracles and blockchain.

Disaster scenarios

Further the most common and the most critical disaster scenarios are provided that should be considered upon production deployment of blockchain.

- 51% attack.
- Disagreement between nodes causing permanent split of blockchain.
- Communication issues: a) failure of consensus because of unavailability or significant desynchronization of validating/mining nodes; b) degrading interaction between nodes in network by introducing multiple fake nodes; c) DDoS attack; d) hanging chain reorganization due to multiple invalid blocks; e) mass exit of validators due to technical or organizational issues.
- Internal logic issues: a) issues in smart contract logic; b) state inconsistency issues; c) consensus algorithm issues halting production of new blocks; d) incorrect pruning issues; e) cryptographic issues causing exposure of private data.

- Governance issues: a) theft of keys used to operate blockchain; b) accidental undesired concentration of voting power within small group of legally related nodes.
- Hacking blockchain logic through external data provided a) via oracles, b) via interchain communication protocols.

Possible issues upon introduction of blockchain

The most important issues caused by introduction of blockchain into business processes may include the following:

a) underresearched legal and governmental limitations or underestimated long-existing business practices that prevent beneficial usage of blockchain;

b) algorithmic and mathematical errors in used libraries/frameworks which may make blockchain non-functional;

c) accidental or malicious placement of private and/or incorrect data into blockchain where they may be immutable;

d) if technical data (like IoT or video surveillance output) will be collected in blockchain its size may grow up to unmanageable one; in countries with insufficient Internet infrastructure even expected blockchain size may cause issues for end-users in remote areas.

Implementation of high-performance blockchains

Custom blockchain that targets the goal of maximum performance should meet the following guideline approximately (only the most important principles are listed):

- use proof-of-stake, delegated proof-of-stake, or BFT consensus;
- use parallelized blocks proposal, validation, and smart contracts execution;
- minimize number of validators with randomizing them for security;
- prefer Wasm over byte-code for smart contracts;
- maximize periodic pruning and caching;
- prefer off-chain execution of logic and off-chain storages with keeping only results of operations and hashes on-chain;
- implement parallel sidechains depending on business purpose;
- optimize interaction between nodes with some reasonable minimal amount;
- batch small transactions.

Legitimate tokens vs so-called “shit-coins”

Speaking correctly, so-called “shit-coins” are not really coins, they are tokens living on some another blockchain, and they have no built-in value, all their value is based on advertisement and speculations. Acceptance of these tokens in exchange for something valuable is based on acceptors’ trust solely.

Typical “shit-coin” may be created with the simplest code of several tens lines in smart contract, or even with running built-in command in some infrastructure. There is no legal framework in such blockchains as Ethereum describing and limiting the procedures of creating and managing “shit-coins”. However, state laws may impose very strict limits on all such ativities.

Usually, all possible offer of such coins is created on initial deployment only; if more coins may be issued later it either should cost reasonable amount of some resources or may be considered as a scam.

If you expect to build legitimate tokens but not “shit-coins”, then you should follow appoximately the guideline provided further:

- base on legitimate blockchain and standardized code for creating tokens to eliminate security risks (e.g. Ethereum, OpenZeppelin ERC-20);
- publish standard events on funds movement between accounts to allow tracking;
- use multiple keys and timelocks for administrator-level operations;
- publish initial offerings to third-party services that will control that no hidden offerings will happen later;
- pass official audits;
- provide maximum legal and technical information on your token online.

Obviously, if you are just a user but not an issuer then checking match to all these guidelines may help you to see difference between market-offered “shit-coins” and legitimate tokens. Also, you may check whether ownership of tokens is highly concentrated or not.

Tools and technologies

Choice of reviewed tools and technologies has been made basing on widespread usage and maximum coverage of fields of application.

FRAMEWORKS FOR BUILDING CUSTOM BLOCKCHAINS		
TOOL	**MAIN LANGUAGES**	**NOTES**
Cosmos SDK	Go smart contracts: Rust/Wasm, Solidity (not on core SDK)	For development of completely independent chains.

Polkadot SDK	Rust smart contracts: Rust/Wasm, Solidity (not on core SDK)	For development of multiple interconnected chains. It has more deep historical connection with Ethereum, but such connection does not mean better interoperability.
Hyperledger Fabric	Go, Node.js, Java	For development of enterprise-level permissioned chains.
Hyperledger Besu	Java smart contracts: Solidity	For development of enterprise-level permissioned Ethereum chains.
Avalanche	Go smart contracts: Solidity	For development of high-performance chains compatible with Ethereum.
PLATFORMS FOR BUILDING BLOCKCHAIN-BASED APPS		
TOOL	**MAIN LANGUAGES**	**NOTES**
Flow	Go (to customize private node); smart contracts: Cadence	For development of consumer-facing chain-based apps (e.g. gaming, finance).
Solana	smart contracts: Rust, C/C++	For development of high-performance chain-based apps.

PLATFORMS TO HOST BLOCKCHAINS	
TOOL	**NOTES**
AWS Managed Blockchain	Support for Hyperledger Fabric, Bitcoin, and Ethereum.
Chainstack	For minimizing complexity in managing nodes, supports 70+ blockchain protocols.
Kaleido	For hosting enterprise-level chains.
SettleMint	For hosting enterprise-level chains. Targeted at simplification of integration of blockchain into business processes.
ETHEREUM-RELATED TOOLS	
Hardhat	Local development environment of smart contracts.
Tenderly	Debug tool and DevOps platform.
Geth, Erigon	Node-running software.
web3.js, ethers.js, wagmi, Web3.py	Libraries to interact with blockchain.
Etherscan	Blockchain explorer
OTHER USEFUL TOOLS	
Slither, Mythril, Echidna, Snyk	Smart contracts security analysis tools.

Infura, Alchemy, QuickNode, Ankr	Presentation of blockchain access as cloud API service without necessity to run own node.
The Graph, SubQuery, Substreams	Indexing and querying frameworks.
Ceramic, DIDKit, Lit Protocol	Decentralized identities' data management tools.
Chainlink	Oracle platform specialized on financial data but may be techically applicable to much more wide range of fields.
DIA	Community-driven open data oracle network.
Wormhole, LayerZero, Axelar, Hyperlane, deBridge	Blockchain interoperability tools.
Blockscout	Blockchain explorer.
Dune, Nansen, Footprint Analytics	Online analytics platforms.

Other options to implement distributed data storages and ledgers

All tools listed further are primarily backend tools targeted to support enterprise-level infrastructure. Depending on your necessities you may find that some of them fit you much more than blockchain-related technologies.

Distributed logs: Apache Kafka.

Distributed SQL databases: CockroachDB, YugabyteDB.

Distributed NoSQL databases: Apache Cassandra, Amazon DynamoDB, MongoDB.

Ledger databases: Amazon QLDB.

The following tools may be considered as alternative for blockchain if you target servicing IoT devices or just extremely large amount of transactions.

Directed acyclic graph tools: IOTA, Nano, Hedera, Constellation, Obyte.

Mathematical foundations

Further I'm explaining some important mathematical foundations of blockchain-based and DAG-based systems to clarify for you the details of their functionality.

FLP impossibility theory vs partial synchrony theory: these theories provide models of functioning for asynchrounous systems that from one side explain that in some cases reaching consensus may be absolutely impossible, and from another side provide basement to implement such systems in a way that in real world allows guaranteed reaching of consensus at some point in time.

Verifiable random function: cryptographic technique that allows owner of private key to produce random unmanipulated output and holders of public key to verify that output was produced correctly; this technique is used by sortition logic to choose validators, block producers etc in fair manner.

Metastable consensus: consensus technique that uses in reaching consensus between not all nodes in the network but just a small randomized sample of nodes; it allows to reach maximum performance in consensus process.

Polynomial commitment: cryptographic technique that allows one party to participate in computation and provide at later stages of computation some proof of correct participation that may be verified by another party without revealing original data of the first party; the approach is used in zero-knowledge proof algorithms.

STARK: the main and the most modern class of protocols of zero-knowledge proof computations; based on public randomness, does not require any secure setup.

Recursive proof: technique that allows to breakdown zero-knowledge proof computation into smaller segments where proof of each segment is based on another one; allows to reach higher performance in large computations.

Verkle trees: more performant and more secure version of Merkle trees (cryptographic struture the blockchain is based on).

Erasure coding: data storage technique that distributes parts of large dataset across many nodes to minimize amount of stored data and reliable recovery under failure conditions.

Markov Chain Monte Carlo tip selection: algorithm used in DAG-based systems to minimize chances to approve malicious transactions.

EIP-1559-style dynamic fees: transaction-pricing mechanism implemented by Ethereum to calculate fee basing on amound of data in previous block; provides more predictable and more user-friendly approach to calculation of fees.

Linkable ring signature: cryptographic technique that allows to sign message anonymously within a group and to verify ownership of signatures; used to minimize double spending and double voting.

Condidential transaction with bulletproof: cryptographic technique to create transactions with hidden amounts but verifiable correctness of transaction.

Zero-knowledge accumulators: cryptographic technique that allows to prove inclusion of value in the set without revealing the set itself.

Latency-bound and bandwidth-hard proof-of-work algorithms: algorithms targeted at democratization of mining with use of general purpose hardware instead of specialized one.

Multiparty signature with treshold: cryptographic technique to allow group of signers to produce single signature (for maximum efficiency, instead of storing multiple signatures) and do it even if some signers are missing but minimal treshold is met.

RSA accumulators with witnesses: cryptographic technique used to whitelist/blacklist sets of coins.

Terminology

I tried to use maximaly simple terminology above to explain multiple technical particularities, but blockchain articles are filled with specific professional lexicon that was invented with a purpose to expand blockchain into multiple fields of applications in the future. Here are some the most important of terms related to legal and business worlds; you may find definitions of these terms and other terms by yourself, I guess:

sovereign chain, parachain, sidechain, rollup;

Layer-0 (L0), Layer-1 (L1), Layer-1.5 (L1.5), Layer-2 (L2), Layer-3 (L3);

oracle;

decentralized autonomous organization (DAO);

slashing economics;

transaction mixer.

Summary

I hope this document demonstrates my general knowledge of blockchain and generally DLT world, I hope you enjoyed reading, and I expect long-term productive work with you.

Thank you for spending your time!