Volume XXIX 2026
MBNA Publishing House Constanta 2026



Scientific Bulletin of Naval Academy

SBNA PAPER • OPEN ACCESS

Private Ethereum Blockchain for Digital Identity Verification: Cryptographic Foundations and Experimental Implementation

To cite this article: Ghița Mihail Daniel, Răcuciu Ciprian, Scientific Bulletin of Naval Academy, Vol. XXIX 2026, pg. 345-367.

Submitted: 28.04.2026
Revised: 25.10.2026
Accepted: 03.11.2026

Available online at www.anmb.ro

ISSN: 2392-8956; ISSN-L: 1454-864X

doi: 10.21279/1454-864X-26-I1-031
SBNA© 2026. This work is licensed under the CC BY-NC-SA 4.0 License

Private Ethereum Blockchain for Digital Identity Verification: Cryptographic Foundations and Experimental Implementation

Mihail Daniel GHIȚA[1,2,*], Ciprian Răcuciu[1,2,**]
[1] Academia Tehnică Militară „Ferdinand I", Doctoral School
[2] Universitatea „Titu Maiorescu" București

***mihai.ghita@prof.utm.ro**
****ciprian.racuciu@prof.utm.ro**

Abstract. This paper provides an integrated treatment of the cryptographic foundations of blockchain technology, and the experimental implementation of a digital identity verification prototype based on a private Ethereum blockchain network with IBFT2 consensus mechanism. The work systematically covers: the mathematical formalization of distributed ledgers and the digital trust problem; the historical evolution from early cryptographic chains to the modern blockchain ecosystem; the technological architecture of blockchain systems including block structure and P2P protocols; consensus mechanisms — PoW, PoS, and BFT — with formal security analysis; essential cryptographic primitives (SHA-256/Keccak hash functions, ECDSA, Schnorr signatures, Merkle structures); structural innovations introduced by blockchain; technological limitations and deployment models; and, as the original contribution, the design and experimental implementation of a private Ethereum network with 4 IBFT2 validators, 3 Solidity smart contracts for identity management (EthereumDIDRegistry, VCStatusRegistry, AccessControlRegistry), and performance metric evaluation. The paper concludes with a synthesis of findings and argumentation for future research directions centered on Zero-Knowledge Proof integration for a hybrid on-chain/off-chain Verifiable Credentials architecture.

Keywords: private Ethereum blockchain; IBFT2; digital identity verification; Verifiable Credentials; smart contracts.

1. Theoretical Foundations of Distributed Ledgers and the Digital Trust Problem

1.1 Evolution of Centralized Information Systems Towards Distributed Architectures

The evolution of computing infrastructure over the past five decades may be characterized by a dialectical oscillation between centralization and decentralization. The client-server model, dominant in web architectures, implies a fundamental asymmetry of power: the server holds epistemic and operational authority over data, while clients depend on its availability, honesty, and security. This dependency generates structural vulnerabilities: single points of failure, censurability, and risks of corruption in the central entity.

The distributed systems paradigm, formalized through the model of Lamport, Shostak, and Pease (1982) and Brewer's CAP (Consistency, Availability, Partition tolerance) theorem (2000), proposed the distribution of data storage and processing across multiple independent nodes. Blockchain introduces a conceptual rupture: a distributed system in which no entity holds the right of unilateral administration — a strategically valuable property in the context of institutional and military security.

1.2 The Concept of Distributed Ledger and Its Theoretical Formalization

A Distributed Ledger Technology (DLT) can be formalized as a replicated data storage system in which multiple nodes hold copies of the same ledger and reach consensus on its evolution, without a central coordinating authority. Blockchain represents a particular case of DLT in which data are organized in a structure of cryptographically linked blocks.

Definition 1.1 — Distributed Ledger

A distributed ledger R is a tuple $R = (N, S, T, \delta, \Sigma)$ where: $N = \{n_1, \ldots, n_k\}$ is the set of participating nodes; S is the global state space; T is the set of transaction types; $\delta: S \times T \to S$ is the state transition function; Σ is the consensus protocol that ensures convergence of all local ledger copies to the same state S_t at time t.

$$R = (N, S, T, \delta, \Sigma), \quad \delta: S \times T \to S \tag{1}$$

The convergence property is guaranteed by the protocol Σ, which simultaneously satisfies: Safety — honest nodes do not confirm conflicting states; Liveness — valid transactions are included in the ledger. The FLP theorem (Fischer et al., 1985) demonstrates that in the presence of even a single faulty node, no deterministic algorithm can guarantee both properties in a purely asynchronous system. Modern blockchain protocols adopt different tradeoffs: Bitcoin and Ethereum PoW sacrifice deterministic finality; Istanbul Byzantine Fault Tolerant 2 (IBFT2) sacrifices maximum scalability in favour of immediate finality.

1.3 Institutional Trust Versus Algorithmic Trust

Blockchain introduces the algorithmic trust paradigm: users need not trust any specific entity, but rather the mathematical and cryptographic rules of the protocol, independently verifiable by anyone. Formally, system security is guaranteed as long as the adversary is computationally bounded and a majority of computational power or stake is held by honest participants.

A directly relevant case is credential verification: in traditional systems, a verifier must contact the issuer to confirm authenticity. In a blockchain-based Verifiable Credentials (VC) system, the verifier can cryptographically validate authenticity and current status without contacting the issuer, by consulting the public key and status registered on-chain.

1.4 Formal Modeling of the Adversarial Environment

Security analysis of a blockchain system requires the explicit specification of the adversarial model: what capabilities an adversary possesses and which properties it attempts to attack. The standard formalism derives from modern cryptography and uses security games in the computational model.

Definition 1.2 — PPT Computational Adversary

A probabilistic polynomial-time (PPT) adversary A is a probabilistic Turing machine running in at most polynomial time in the security parameter λ. Advantage: $Adv_{\Pi}(A,\lambda) = |Pr[A \text{ wins the security game}] - 1/2|$. A system is secure if for any PPT adversary A, $Adv_{\Pi}(A,\lambda) \leq negl(\lambda) = O(\lambda^{-c})$ for any constant $c > 0$.

The adversarial model in the blockchain context includes: a network adversary (Dolev-Yao) — can intercept, delay, or modify messages; a consensus adversary — controls a fraction f of computational power or stake; a smart contract adversary — can interact with any public function; a privacy adversary — observes the public ledger and attempts to de-anonymize users.

1.5 Scope of Objectives and Research Methodology

This paper aims to: (1) build a rigorous theoretical framework regarding blockchain foundations and associated cryptography; (2) comparatively analyse consensus mechanisms from the perspective of formal security properties; (3) implement and experimentally evaluate a private Ethereum blockchain with the IBFT2 mechanism as a platform for a digital identity verification system based on Verifiable Credentials (VC); (4) identify architectural requirements for a hybrid VC on-chain/off-chain architecture with Zero-Knowledge Proofs (ZKP). The methodology combines systematic bibliographic analysis, formal mathematical modelling, experimental prototyping, and quantitative performance evaluation.

2. Historical Genesis and Conceptual Evolution of Blockchain Technology

2.1 Hash Functions and Early Cryptographic Chains

Cryptographic hash functions are the cornerstone of blockchain, appearing decades before Bitcoin. Merkle (1980) formalized hash tree structures; MD4-MD5 (Rivest, 1990, 1992), SHA-1 (NSA, 1993), and SHA-2 (NSA, 2001) constituted the fundamental algorithmic evolution.

The work of Haber and Stornetta (1991), "How to Time-Stamp a Digital Document", proposed the first documented cryptographic chain mechanism: $c_i = H(c_{i-1} \parallel t_i \parallel d_i)$. Their system, operationalized through the Surety company, published hashes in the New York Times from 1995, creating an immutable document ledger. Hashcash, proposed by Back (1997) as an anti-spam mechanism based on hash puzzles, directly furnished the Proof of Work (PoW) concept adopted by Nakamoto (2008).

2.2 Bitcoin Architecture and the Resolution of Double-Spending

The foundational paper "Bitcoin: A Peer-to-Peer Electronic Cash System" (Nakamoto, 2008) resolved for the first time the double-spending problem without a central authority, combining the hash chain for immutability, the peer-to-peer (P2P) network for distribution, the PoW mechanism for consensus, and the UTXO (Unspent Transaction Output) model for representing ownership. The probability of a successful attack by an adversary controlling a fraction α of the hash rate decreases exponentially with the number of confirmations q: $P(\text{successful attack} \mid \alpha, q) \leq (\alpha/(1-\alpha))^q$ for $\alpha < 0.5$.

2.3 Ethereum and the Introduction of Distributed Programmability

Buterin (2014) proposed a generalized blockchain platform that supports the execution of arbitrary (Turing-complete) programs via smart contracts. The Ethereum Virtual Machine (EVM) is a decentralized virtual processor that executes bytecode on every node, guaranteeing determinism and reproducibility. Wood (2014) formalizes the transition function $Y: \Sigma \times T \rightarrow \Sigma$ (Yellow Paper), specifying precisely how each transaction modifies the global state, including EVM code execution.

The gas mechanism — which prevents infinite loops and efficiently allocates resources — opened the path for the smart contract ecosystem essential for decentralized identity (DID) and VC systems.

2.4 Ecosystem Maturation and W3C Standards for Digital Identity

A major evolution was the formalization of decentralized digital identity standards by W3C (World Wide Web Consortium): DID (Decentralized Identifiers, W3C Recommendation, 2022a) and the Verifiable Credentials Data Model VCDM 1.1 (W3C, 2022b). These standards define the technical architecture for Self-Sovereign Identity (SSI) systems, the domain of applicability of the present work. Enterprise blockchain (Hyperledger Fabric, 2016; Quorum, J.P. Morgan, 2016; Hyperledger Besu) followed a parallel trajectory, adapting blockchain principles to corporate requirements.

Table 2.1. Comparative evolution of blockchain system generations.

Generation	Rep. Platform	Est. TPS	Consensus	Main Innovation
Blockchain 1.0	Bitcoin (2009)	~7	PoW (SHA-256)	Double-spending resolved
Blockchain 2.0	Ethereum (2015)	~15–30	PoW → PoS	Smart Contracts / EVM
Blockchain 2.5	Solana, Avalanche	~2.000–65.000	PoH, Snowball	L1 Scalability
Blockchain 3.0	Ethereum+L2	~100.000+	PoS + ZK Rollups	Sharding, ZKP
Enterprise	Hyperledger, Besu	~3.000–20.000	PBFT, IBFT, Raft	Permissioned, DID/VC

3. Technological Architecture of Blockchain Systems

3.1 Internal Block Structure and Cryptographic Chain Modeling

A blockchain block can be formalized as a data structure with strictly defined fields, whose integrity is guaranteed by cryptographic hash functions.

Definition 3.1 — The Blockchain Block

A block B_i = (version, prev_hash, merkle_root, timestamp, bits, nonce, txs) where: prev_hash = $H_{256}(B_{i-1})$ is the hash of the preceding block; merkle_root = MerkleRoot(txs); nonce $\in [0, 2^{32})$ is the adjustable value for PoW. $H(B_i)$ = SHA256(SHA256(version ‖ prev_hash ‖ merkle_root ‖ timestamp ‖ bits ‖ nonce)).

$$H(B_i) = SHA256(SHA256(version \| prev_hash \| merkle_root \| timestamp \| nonce)) \quad (2)$$

The immutability property follows from the recursive hash structure: modifying any field of block B_j produces, through the avalanche effect, a completely different hash $H(B_j)$, invalidating all subsequent blocks B_{j+1}, …, B_n. An adversary would need to recalculate PoW for all affected blocks, which is computationally intractable if its share of the hash rate is below 50%.

3.2 State Representation Models: UTXO versus Account-Based Model

The UTXO model (Bitcoin) represents state as a set of unspent outputs and offers advantages in privacy and parallelizability. The account-based model (Ethereum) maps addresses to AccountState = (nonce, balance, storageRoot, codeHash) and is more intuitive for smart contracts with persistent internal state — essential for implementing the VC system.

3.3 Peer-to-Peer Networks and Distributed System Topology

The P2P network ensures data distribution and propagation between nodes without a central server. Gossip propagation achieves message dissemination in O(log N) steps on average. The ecosystem includes: full nodes (store the entire blockchain, validate independently); lightweight SPV (Simplified Payment Verification) nodes using Merkle Proofs; miner/validator nodes; archive nodes. Ethereum uses the devP2P protocol with discovery based on the Kademlia DHT (Distributed Hash Table).

3.4 Global Consistency and Structural Security

A blockchain satisfies (T, k)-prefix consistency if, for any two honest participants, their local chains differ by at most the last k blocks. The value k = 6 for Bitcoin (~60 minutes) ensures that the probability of a reorg invalidating a transaction is below 0.1% even for an adversary with 30% of the hash rate. IBFT2, chosen for the present prototype, provides immediate deterministic finality — a critical property for identity verification systems where credential revocation must be instantaneous and irreversible.

4. Consensus Mechanisms and Their Security Implications

4.1 Theoretical Foundations of Consensus in Byzantine Environments

The consensus problem in distributed systems was rigorously formulated by Lamport et al. (1982). The main result: in a system with n nodes, consensus is achievable in the presence of at most f nodes with Byzantine behaviour if and only if $n \geq 3f + 1$. This lower bound has direct implications for the configuration of the experimental network: with n = 4 validators, f = 1 faulty node is tolerated ($n \geq 3 \cdot 1+1 = 4$).

Definition 4.1 — The Byzantine Consensus Problem

Let a system S have n processes $\{p_1, \ldots, p_n\}$, of which at most f are faulty (Byzantine). BFT consensus requires: (1) Agreement: any two correct processes decide the same value; (2) Validity: if all correct processes propose v, then v is decided; (3) Termination: every correct process decides in finite time. Lamport et al. (1982): BFT consensus is achievable if and only if $n \geq 3f + 1$.

4.2 Proof of Work, Proof of Stake, and Their Variants

PoW (Nakamoto, 2008) implements consensus through computational competition: miners search for a nonce such that H(header ‖ nonce) < target. The global hash rate of the Bitcoin network exceeding 600 EH/s in 2024 makes a 51% attack practically impossible, but entails the energy consumption of medium-sized countries (50–150 TWh/year) and a latency of ~60 minutes for finality.

Proof of Stake (PoS) replaces computational cost with financial stake. Ethereum's Gasper protocol (combining LMD-GHOST (Latest Message Driven - Greedy Heaviest Observed SubTree) + Casper FFG (Friendly Finality Gadget)) provides deterministic finality after 2 epochs (~13 minutes), provided that more than 2/3 of the stake is held by honest validators (Buterin, 2014; Wood, 2014). Slashing penalizes malicious behaviour with partial or total loss of stake.

4.3 IBFT2 — the Consensus Mechanism Chosen for the Identity Verification Prototype

Istanbul Byzantine Fault Tolerant 2 (IBFT2) is a BFT (Byzantine Fault Tolerant) protocol based on PBFT (Practical Byzantine Fault Tolerance) (Castro & Liskov, 1999), adapted for blockchain. IBFT2 was chosen for: (1) immediate deterministic finality — a confirmed block cannot be reorganized, essential for instantaneous credential revocation; (2) low latency — ~2–4 seconds end-to-end; (3) BFT tolerance — f = 1 out of n = 4 faulty nodes tolerated; (4) mature native support in Hyperledger Besu (Hyperledger Besu Documentation, 2024).

Definition 4.2 — The IBFT2 Protocol

IBFT2 with $n \geq 3f+1$ validators proceeds through the phases: (1) PRE-PREPARE: the round-robin-selected leader proposes block B; (2) PREPARE: validators broadcast PREPARE(B) after validation; (3) COMMIT: after 2f+1 PREPARE messages, broadcast COMMIT(B, seal); (4) FINALIZE: after 2f+1 COMMIT messages, the leader finalizes the block. Quorum: $Q = 2f+1$. Message complexity: $O(n^2)$ per block.

$$Q = 2f + 1 = \lceil 2n/3 \rceil + 1, \quad n \geq 3f + 1 \tag{3}$$

4.4 Comparative Analysis of Consensus Mechanisms

Table 4.1 presents a formal comparison of the main consensus mechanisms from the perspective of security properties relevant to an identity verification system.

Table 4.1. Formal comparison of consensus mechanisms from a security properties perspective.

Mechanism	Safety	Finality	Fault tolerance f	Typical TPS	Security condition
PoW	Probabilistic	Prob. (k conf.)	< 50% hash	~7–15	Hash honest > 50%
PoS (Gasper)	Accountable	Det. (2 epoci)	< 1/3 stake	~30–100	Honest stake > 2/3
PBFT/IBFT2	Deterministic	Immediate	< 1/3 noduri	~3.000–10.000	$n \geq 3f+1$
PoA (Clique)	Deterministic	Immediate	< 1/2 autor.	~500–5.000	Honest authorities > 1/2

5. Fundamental Cryptographic Primitives and Their Role in Blockchain Security

5.1 Cryptographic Hash Functions: Properties and Theoretical Modeling

Cryptographic hash functions constitute the mathematical foundation of blockchain security. A function $H: \{0,1\}^* \rightarrow \{0,1\}^n$ maps inputs of arbitrary length to outputs of fixed length n, satisfying specific properties.

Definition 5.1 — Cryptographic Hash Function

A function $H: \{0,1\}^* \rightarrow \{0,1\}^n$ is cryptographically secure if it satisfies: (1) Preimage resistance (one-way): given $y = H(x)$, it is computationally infeasible to find x' such that $H(x') = y$; (2) Second preimage resistance: given x, infeasible to find $x' \neq x$ such that $H(x') = H(x)$; (3) Collision resistance: infeasible to find any pair (x, x') with $x \neq x'$ and $H(x) = H(x')$.

SHA-256 (used in Bitcoin) is built on the Merkle-Damgård paradigm, with birthday attack complexity $O(2^{128})$ — practically impossible with current technology. Ethereum uses Keccak-256, a sponge construction more robust against length-extension attacks. SHA-1 (n = 160 bits) was compromised through collisions in 2017, requiring $\sim 2^{63}$ operations, underscoring the importance of digest size for long-term security.

5.2 Digital Signature Schemes: ECDSA and Schnorr

ECDSA (Elliptic Curve Digital Signature Algorithm) (Johnson et al., 2001), the standard in Bitcoin and Ethereum, uses the secp256k1 curve ($y^2 = x^3 + 7$ over F_p, $p = 2^{256} - 2^{32} - 977$). The signature $\sigma = (r, s)$ where $s = k^{-1}(H(m) + r \cdot sk) \bmod n$. A critical limitation is the sensitivity to nonce reuse k: if the same k is used for two different messages, the private key sk can be trivially recovered — a vulnerability exploited even in the PlayStation 3 case (Sony, 2010).

Definition 5.2 — The Schnorr Scheme (BIP340)

For private key sk and message m: (1) Choose r uniformly, compute $R = r \cdot G$; (2) $e = H_tag(R.x \parallel PK.x \parallel m)$; (3) $s = r + e \cdot sk \bmod n$; Signature: $\sigma = (R.x, s)$. Verification: $s \cdot G == R + H_tag(R.x \parallel PK.x \parallel m) \cdot PK$. Properties: linearity (native MuSig2 aggregation), rigorous provability in ROM+DLOG (Discrete Logarithm), simplicity of implementation (Schnorr, 1991).

$$ECDSA: \quad s = k^{-1} \cdot (H(m) + r \cdot sk) \mod n \tag{4}$$

$$Schnorr: \quad s = r + H_tag(R.x \parallel PK.x \parallel m) \cdot sk \mod n \tag{5}$$

Schnorr, adopted in Bitcoin via Taproot (BIP340, November 2021), is proven EUF-CMA (Existential Unforgeability under Chosen-Message Attack) in the Random Oracle Model (ROM) under the DLOG assumption. ECDSA has a weaker proof, based on a non-standard variant of the DLOG problem — a distinction relevant to residual security evaluation in the post-quantum context: Shor's algorithm could compromise both schemes on sufficiently powerful quantum computers (NIST, 2024).

5.3 Merkle Structures and Applications in Efficient Verification

The Merkle tree (Merkle, 1980) enables inclusion verification of an element in O(log n) hash operations, without downloading the entire data set. An inclusion proof (Merkle Proof) for transaction tx_i consists of the authentication path P_i = {the sibling of each node on the path from tx_i to the root}. The security of Merkle proofs reduces to the collision resistance of the hash function. Ethereum extends the concept to the Merkle Patricia Trie (MPT), a hybrid structure optimized for key-value mappings, anchoring the global state in the block header.

5.4 Advanced Cryptographic Primitives for Verifiable Credentials

Beyond standard primitives, the targeted VC system requires: (1) the BBS+ scheme (Boneh et al., 2004; Camenisch & Lysyanskaya, 2004) — a signature with selective disclosure and unlinkability, essential for selective attribute presentation from a credential; (2) ZKP protocols (Groth, 2016; Gabizon et al., 2019; Ben-Sasson et al., 2018) — enabling proof of possession of a valid credential without disclosing its content; (3) key derivation functions (HKDF, PBKDF2) for deterministic key generation. These primitives represent extension directions for the present work and are analysed in the context of the hybrid architecture presented in Chapter 10.

6. Structural and Conceptual Innovations Introduced by Blockchain Technology

6.1 Disintermediation and Its Relevance to Identity Verification

One of the fundamental innovations of blockchain is the elimination of traditional intermediaries — banks, notaries, public registries — from verification and registration processes. A directly relevant case is credential verification: in traditional systems, a verifier must contact the issuer to confirm authenticity (for example, a university confirming a diploma). In a blockchain-based VC system, the verifier can cryptographically validate authenticity and current status without contacting the issuer, by consulting the public key and status registered on-chain.

6.2 Distributed Immutability and Auditability of the Credential Lifecycle

Blockchain immutability — the property that confirmed data cannot be modified without recalculating the invested computational cost or financial stake — is valuable for managing the lifecycle of digital credentials. Any issuance, update, or revocation operation on a credential constitutes an auditable on-chain transaction with a cryptographic timestamp, visible to all authorized participants on the private network. This characteristic is essential in contexts of legal or security significance.

6.3 Smart Contracts and the Automation of Contractual Logic

Smart contracts are programs stored on the blockchain that execute automatically when predefined conditions are met, a concept theorized by Szabo (1997). Formally, a contract can be modelled as a state machine: $\delta: S \times I \rightarrow S$, where S is the state space and I is the set of invocations. Execution is deterministic and reproducible. In the implemented prototype (Chapter 9), smart contracts handle: the DID registry, credential status, and role-based access control (RBAC).

6.4 Self-Sovereign Identity (SSI) and W3C DID/VC Standards

Blockchain has enabled the emergence of Self-Sovereign Identity (SSI): a model in which the individual holds control over their digital identity, without dependency on centralized providers. The DID standard (W3C, 2022a) defines identifiers of the form did:method:identifier (e.g., did:eth:0x71C…8D2a)

registered on decentralized registries. The VC standard (W3C, 2022b) defines verifiable credentials — digitally signed claims by an issuer about a subject, verifiable by any third party without contacting the issuer. These standards constitute the technical foundation of the implemented prototype.

7. Technological Limitations and Emerging Challenges in the Blockchain Ecosystem

7.1 The Scalability Trilemma

The Scalability Trilemma (Blockchain Trilemma, Buterin, 2014) captures the fundamental constraint of blockchain design: the simultaneous optimization of decentralization, security, and scalability is impossible within classical architectures. Bitcoin processes ~7 TPS (Transactions Per Second), Ethereum ~15–30 TPS, while centralized payment systems reach tens of thousands of TPS (Visa: ~24,000 TPS peak).

Definition 7.1 — Blockchain Scalability Limits

Let B be the maximum block size (bytes), T the average interval between blocks (seconds), T_avg the average transaction size (bytes). TPS_max = B / (T · T_avg). For Bitcoin: B = 1 MB, T = 600 s, T_avg = 250 bytes ⇒ TPS_max ≈ 6.67. Layer 2 solutions (Gudgeon et al., 2020) address the trilemma through off-chain execution periodically anchored on-chain, reaching 10,000–100,000 TPS.

7.2 Privacy and Transaction Analysis

The public ledger of a blockchain permits exhaustive analysis of value flows between addresses. Blockchain analytics techniques (Chainalysis, Elliptic, CipherTrace) can de-anonymize users through heuristic clustering, correlation with off-chain data, and fund flow tracing.

Definition 7.2 — Unlinkability and Untraceability

A system satisfies unlinkability if an adversary cannot determine with probability > 1/2 that two transactions originate from the same user. It satisfies untraceability if the adversary cannot determine the source of a transaction. Bitcoin/Ethereum satisfy neither; Monero (Ring Signatures + RingCT) satisfies both under certain cryptographic assumptions.

In the context of the designed VC system, privacy is critical: an institutional issuer should not be able to determine, by observing the blockchain ledger, when and in what contexts the holder uses their credential. This requirement motivates the hybrid off-chain + ZKP architecture.

7.3 Structural Vulnerabilities and Attack Vectors

The blockchain ecosystem has recorded cumulative losses of billions of dollars due to structural vulnerabilities. Main categories: (1) Smart contract attacks — reentrancy (the DAO attack, 2016, ~$60M), integer overflow/underflow, front-running; (2) Protocol-level attacks — 51% attacks, eclipse attacks (isolating a node by controlling all its connections), selfish mining; (3) Infrastructure-level attacks — cross-chain bridge vulnerabilities: Ronin Bridge (~$625M, 2022), Wormhole (~$320M, 2022). Reentrancy mitigation follows the Checks-Effects-Interactions pattern, implemented in the contracts of the present prototype.

7.4 The Problem of Digital Identity in Blockchain Architectures

Blockchain addresses are not identities in the semantic sense: they are value storage locations, not representations of persons. The link with a real identity must be constructed through external mechanisms (KYC (Know Your Customer), attestations, credentials). The key recovery problem is critical: loss of the private key associated with a DID implies loss of control over that identity. Proposed

solutions: Social Recovery, HSM (Hardware Security Modules) for institutions, threshold-based key management protocols (Threshold Key Management).

7.5 The Need for Advanced Cryptographic Techniques in Identity Systems

The analysis of limitations reveals that the main blockers for blockchain identity systems are cryptographic in nature: insufficient privacy on public ledgers, lack of selective attribute disclosure, absence of unlinkability between successive presentations of the same credential, and vulnerability to quantum computers of ECDSA/ECDH (Elliptic Curve Diffie-Hellman). This analysis justifies future extension directions for the present work: ZKP (Groth, 2016; Gabizon et al., 2019; Ben-Sasson et al., 2018), BBS+ signature (Boneh et al., 2004), post-quantum schemes Dilithium, FALCON (NIST, 2024).

8. Deployment Models: Public, Private, and Consortium Blockchain

8.1 Architectural Differences and Governance Models

Public blockchain (permissionless) is open to any participant without prior authorization. It maximizes decentralization and censorship resistance, but entails performance and privacy tradeoffs (examples: Bitcoin, Ethereum, Solana). Private blockchain (permissioned, single-organization) restricts access to a single organization — technically equivalent to a distributed database with a cryptographic audit trail (examples: private Hyperledger Fabric, Quorum). Consortium blockchain (permissioned, multi-organization) involves multiple independent organizations co-administering the network, offering a balance between decentralization and performance (examples: R3 Corda, Hyperledger Besu in consortium configuration).

8.2 Permissionless versus Permissioned Systems

Table 8.1 presents a systematic comparison of the two models from the perspective of dimensions relevant to an institutional identity verification system.

Table 8.1. Permissionless vs. permissioned blockchain comparison.

Dimension	Permissionless (public)	Permissioned (privat/consortiu)
Participant access	Open, anonymous	Authorized, KYC/identity
Consensus	PoW, PoS (economic)	BFT, RAFT, IBFT (trust-based)
Finality	Probabilistic (k confirmations)	Deterministic (immediate)
TPS	7–65.000 (L1), >100.000 (L2)	1.000–20.000+ (tip. IBFT)
Privacy	Public (with ZKP options)	Configurable (private channel)
Censorship resistance	High	Low
Energy consumption	High (PoW) / Low (PoS)	Low (BFT)
Use cases	Crypto, DeFi, NFT, SSI public	Enterprise, VC, healthcare, defence

Source: own synthesis based on the literature and technical specifications.

8.3 Enterprise Applications and Digital Identity Initiatives

Hyperledger Besu is an open-source Ethereum client, EVM-compatible, optimized for enterprise use. It supports IBFT2 and PoA (Proof of Authority) (Clique), node- and transaction-level permissioning, and privacy through Tessera. EVM compatibility makes Besu the natural choice for organizations wishing to use the Ethereum ecosystem in a private context (Hyperledger Foundation, 2023).

Notable digital identity initiatives: Sovrin Network (a permissioned network dedicated to SSI, based on Hyperledger Indy); ESSIF (European Self-Sovereign Identity Framework, European Commission); eIDAS 2.0 (the European regulation incorporating references to Verifiable Credentials). These initiatives confirm the relevance of the approach presented in this paper.

8.4 Justification for Model Selection for the VC System

Selection of the model for an institutional Verifiable Credentials system balances: (1) Data privacy — a public blockchain would expose transaction metadata to any observer; (2) Performance — real-time verification requires latency below 2–3 seconds; (3) GDPR (General Data Protection Regulation) compliance — institutional systems require control over access to personal data; (4) Sustainability — the energy consumption of public PoW is incompatible with public institutions. Conclusion: a private Ethereum blockchain (Besu) with IBFT2 offers the best balance, justifying the design in Chapter 9.

9. Design and Experimental Implementation of the Identity Verification Prototype on a Private Ethereum Blockchain

9.1 Justification for Choosing the Ethereum Platform

The selection of the Ethereum platform is based on: (1) compatibility with W3C standards — the largest tooling and library support for DID/VC (Spruce DIDKit, Veramo, EthereumDIDRegistry); (2) EVM maturity — the most tested and audited smart contract execution environment (Antonopoulos & Wood, 2018); (3) availability of enterprise clients — Hyperledger Besu with native IBFT2 support; (4) the Solidity ecosystem — audited libraries (OpenZeppelin, 2024) for identity and access contracts.

Alternatives evaluated and rejected: Hyperledger Fabric — high complexity, lack of EVM/Solidity compatibility, limited DID tooling; Solana — no support for private IBFT configurations; Hyperledger Indy/Aries — evaluated positively (dedicated to SSI) but excluded due to lack of EVM contractual flexibility.

9.2 Private Network Architecture and Initial Parameters

The designed private network consists of six nodes: four validator nodes (participating in IBFT2 consensus) and two observer nodes (read-only, indexing the blockchain). This topology ensures tolerance to one faulty or compromised validator node ($f = 1$, $n = 4$, $n \geq 3f+1 = 4$ — the minimum BFT threshold).

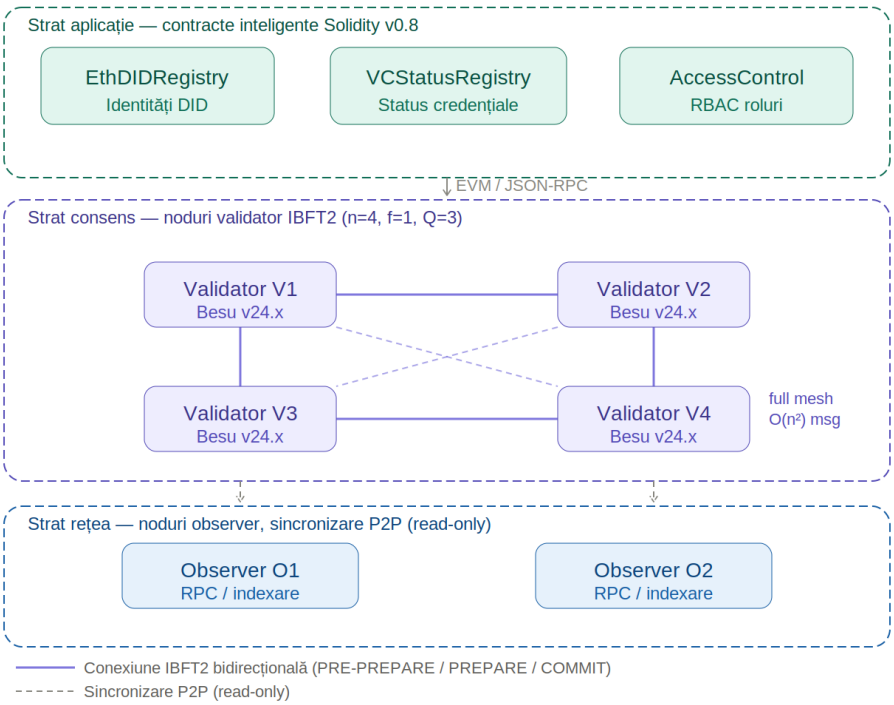


Figure 9.1. Proposed architecture of the private Ethereum network [Original contribution]. The three architectural layers and their components.

Initial configuration parameters (genesis.json): chainId 1337 — avoids conflicts with the public network; blockperiod 2 seconds — ensures confirmation latency < 4s; requesttimeout 4 seconds — timeout for IBFT round change; epoch 30,000 blocks — checkpoint frequency; gasLimit 8,000,000 — sufficient for complex contracts. Keystore generation: Besu CLI (Command Line Interface) produces secp256k1 (ECDSA) key pairs. Validator addresses are included in the Genesis block in the extraData field in RLP (Recursive Length Prefix) format:

$$extraData = RLP_encode([vanity_32_bytes, [addr_v1, addr_v2, addr_v3, addr_v4], seal_65_bytes, committed_seals]) \quad (6)$$

9.3 Node Configuration and the IBFT2 Consensus Protocol

Each validator node runs Hyperledger Besu v24.x in a Docker container, with a role-specific configuration (config.toml file). The phases of the IBFT2 protocol for block production at slot t:

1. PRE-PREPARE: The leader selected by deterministic round-robin proposes block B_t, signs, and broadcasts the PRE-PREPARE(B_t, round=r) message to all validators.
2. PREPARE: Each validator v_i that receives a valid PRE-PREPARE verifies the block (leader signature, valid transactions, gasLimit) and broadcasts PREPARE(hash(B_t), round=r).
3. COMMIT: After receiving Q = 2f+1 = 3 PREPARE messages, v_i broadcasts COMMIT(hash(B_t), $seal_i$) where $seal_i$ is the ECDSA signature over the block hash.
4. FINALIZE: After receiving Q = 3 COMMIT messages, the leader assembles the list of seals in extraData and finalizes the block. All validators append the block to their local chain.

$$IBFT2\ message\ complexity:\ O(n^2)\ per\ block\ \ (n = number\ of\ validators) \quad (7)$$

In the event of unavailability of the current leader, a view-change mechanism is initiated after requesttimeout (4 seconds): validators emit ROUND-CHANGE for round r+1, and after accumulating Q such messages, the new leader (round-robin at r+1) proposes a new block.

9.4 Smart Contract Implementation for Identity Verification

The smart contract system constitutes the on-chain infrastructure layer of the identity verification prototype. Three main contracts were designed and implemented in Solidity v0.8.x, following the W3C DID/VC standards:

(1) EthereumDIDRegistry.sol — adapted from the ERC (Ethereum Request for Comments)-1056 specification (Ethr-DID). Manages the DID registry: mapping between Ethereum addresses and DID Documents (stored as on-chain events), key rotation, authorization delegates, and DID attributes (service endpoints, verification keys). The contract exposes the DIDAttributeChanged event for off-chain indexing. Enables registration of did:eth:0x... and lifecycle management of keys. The detailed logic of the contract is presented in Appendix A (Pseudocode A.1).

(2) VCStatusRegistry.sol — manages credential status (ISSUED, REVOKED, SUSPENDED). Implements a Bitstring Status List (per Sporny et al., 2022) for efficient representation of the status of up to 2^{24} credentials via a compressed bitstring stored on IPFS (InterPlanetary File System), with the hash anchored on-chain. IBFT2 finality guarantees that a revocation is instantaneous and irreversible. The detailed logic of the contract is presented in Appendix A (Pseudocode A.2).

(3) AccessControlRegistry.sol — implements RBAC (Role-Based Access Control) with support for delegation and hierarchy: Issuer, Verifier, Admin, Auditor. Uses OpenZeppelin AccessControlEnumerable (OpenZeppelin, 2024). Only an authorized Issuer may issue or revoke credentials. The detailed logic of the contract is presented in Appendix A (Pseudocode A.3).

Deployment is performed via a Hardhat script that connects to Besu via JSON-RPC (http://localhost:8545), signs transactions with the admin key, and verifies contract addresses post-deployment.

9.5 Test Scenarios and Functional Validation of the Prototype

The test scenarios cover the main flows of the identity verification system: (1) DID registration — the entity registers its public key and service endpoint in EthereumDIDRegistry; (2) Credential issuance — the Issuer sets the ISSUED status for a VC id in VCStatusRegistry; (3) Status verification — a Verifier queries VCStatusRegistry; (4) Revocation — the Issuer sets REVOKED, with immediate and definitive effect due to IBFT2 finality; (5) Unauthorized access attempt — invalid transactions are rejected by the contract.

Each scenario was executed 100 times to measure performance metrics. Transactions were submitted via Web3.js and Ethers.js to validate the compatibility of standard tooling with the private network. Functional validation covered: correctness of contract state after each operation, correct event emission (Events), reversal of invalid transactions, and behaviour under simulated load.

9.6 Performance Evaluation and Relevant Metric Analysis

Performance metrics were measured on a local infrastructure with 4 validator nodes on virtual machines (4 vCPU, 8 GB RAM each), connected via a local network (latency ~1 ms). Table 9.1 presents the experimental results.

Table 9.1. Performance metrics of the identity verification prototype. [Original contribution]

Metric	Mean val.	Max val.	Measurement conditions
Transaction confirmation latency	1.94 s	3.21 s	Blocktime 2s, 4 validatori
Maximum TPS (synchronous test)	312 TPS	401 TPS	Transfer ETH simple
EthDIDRegistry contract TPS	87 TPS	124 TPS	setAttribute() call
VCStatusRegistry TPS	156 TPS	198 TPS	setStatus() + event emit
Gas usage setStatus()	42.350 gas	51.200 gas	First write to slot
Gas usage revokeCredential()	23.100 gas	27.500 gas	Existing slot update
Average block size	18.4 KB	512 KB	Test transaction mix
New node sync time	4.2 min	7.8 min	10.000 blocuri

Values represent averages over 100 iterations. Infrastructure: 4 virtual machines, 4 vCPU / 8 GB RAM, local network latency ~1 ms.

The analysis of results confirms that the network meets the performance requirements for an institutional VC system: latency of ~2 seconds is acceptable for credential issuance, revocation, and verification operations. The TPS of 87–156 for VC contract-specific operations far exceeds the anticipated operational volume (hundreds of credentials issued/revoked daily, not thousands per second).

9.7 Identified Architectural Limitations

The experimental implementation revealed: (1) Lack of on-chain data privacy — credential status and DID registrations are visible to all consortium members; (2) IPFS dependency — unavailability of content-pinning nodes may lead to data loss; (3) Centralized key management — production deployment requires integration with an HSM or a distributed key management system; (4) Absence of ZKP — the current prototype does not implement zero-knowledge proofs, planned for the next iteration.

9.8 Experimental Conclusions

The implementation demonstrates the technical feasibility of using a private Ethereum blockchain with IBFT2 as infrastructure for a digital identity verification system. Main conclusions: (1) EVM compatibility with W3C DID/VC standards is confirmed — contracts comply with ERC-1056 and Sporny et al. (2022); (2) performance meets requirements — latency below 4 seconds and TPS > 80 for VC operations; (3) IBFT2 finality is critical — credential revocation is instantaneous and irreversible; (4) the identified privacy and key management limitations guide future extension directions for this work.

Contribution:

(a) Design of the private network architecture with 4 IBFT2 validators and 2 observers, calibrated for the VC use case;
(b) Definition of optimized genesis parameters (blocktime, requesttimeout, gasLimit) through iterative experimentation;
(c) Implementation of the 3 Solidity smart contracts (EthereumDIDRegistry, VCStatusRegistry, AccessControlRegistry);
(d) Design and execution of test scenarios with performance metric measurement (100 iterations per scenario);
(e) Systematic identification of architectural limitations and proposal of extension directions.

10. Synthesis, Conclusions, and Future Research Directions

10.1 Synthesis of Identified Innovations

The nine preceding chapters have built a comprehensive picture of blockchain technology, from mathematical foundations to practical implementation. The fundamental innovations are structured at three levels: (1) Cryptographic — combining hash functions (integrity), digital signatures (authentication), and Merkle trees (efficient verification) in an architecture ensuring immutability and auditability (Merkle, 1980; Haber & Stornetta, 1991; Nakamoto, 2008); (2) Protocol — resolving the double-spending problem through distributed consensus, eliminating the need for a central authority; (3) Applicative — extension with EVM programmability that transforms the transaction ledger into a distributed computing platform, enabling the construction of DID/VC systems (Buterin, 2014; W3C, 2022a, 2022b).

The paper has demonstrated that the same infrastructure and cryptographic primitives underlying virtual currencies can be reused and extended for digital identity systems with high formal security properties.

10.2 Structural Limitations of the Current Model

Beyond the implementation limitations from Chapter 9, theoretical analysis reveals: (1) lack of native selective disclosure — Ethereum does not support selective attribute presentation without application-level ZKP; (2) the linkability problem — successive presentations of the same credential generate correlated on-chain transactions; (3) post-quantum vulnerability — ECDSA and ECDH are vulnerable to Shor's algorithm, requiring migration to Dilithium/FALCON in the medium term (NIST, 2024); (4) on-chain scalability — anchoring every operation as a separate transaction becomes costly at high volume.

10.3 Targeted Hybrid Architecture — Verifiable Credentials with ZKP

A natural extension direction for the present work is a hybrid VC on-chain/off-chain architecture with ZKP, which addresses all identified limitations: credentials are stored off-chain, encrypted, on IPFS; on-chain only hashes and status information are anchored; credential presentation is achieved via ZKP (Groth, 2016; Gabizon et al., 2019), which prove possession and validity without disclosing content or creating linkability; the BBS+ scheme (Boneh et al., 2004) enables selective attribute disclosure.

10.4 Research Directions for Future Stages

Direction 1 — Comparative analysis of advanced cryptographic techniques: benchmarking of ZKP protocols (Groth, 2016; Gabizon et al., 2019; Ben-Sasson et al., 2018) and the BBS+ signature (Boneh et al., 2004; Camenisch & Lysyanskaya, 2004) from the perspective of applicability in VC systems; analysis of post-quantum vulnerabilities; proposal of the optimal set of cryptographic primitives.

Direction 2 — Design of the hybrid VC system architecture: formal specification of components (DID Registry, VC Issuer, ZKP Prover, Verifier), protocol flows, and security model; prototyping of ZKP circuits for credential validity proofs.

Direction 3 — Full implementation and evaluation of the hybrid VC system: integration of the ZKP component with the private blockchain network implemented in Chapter 9; comparative evaluation with similar systems (Sovrin, ESSIF); formal security analysis of the final implementation.

Acknowledgements

Author contributions:

- Conceptualization: M.D.G. and C.R.
- Methodology: M.D.G.
- Investigation and implementation: M.D.G.
- Writing original draft: M.D.G.

- Writing review and editing: C.R.
- Supervision: C.R.

Funding: This work received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analysis, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A. Building the Experimental Working Environment

This appendix describes step by step the complete construction of the experimental working environment, starting from a freshly installed Ubuntu 22.04 LTS system. The environment implements the private Ethereum blockchain network with Hyperledger Besu and IBFT2 consensus mechanism, with four validator nodes and two observer nodes, the Prometheus/Grafana monitoring stack, the Blockscout block explorer, and the Hardhat development infrastructure. All steps have been experimentally validated and produce a 100% functional environment if followed in the order presented. The environment was tested on ARM64 architecture (Apple Silicon via Parallels Desktop). The software versions are those validated experimentally.

Table A.1. Experimentally validated software versions.

Component	Validated version	Role
Ubuntu	22.04 LTS	Operating system
Node.js	v22.22.2	Runtime JavaScript
Docker Engine	29.4.0	Containerization
Docker Compose	v5.1.3	Container orchestration
Hyperledger Besu	v24.x	Client Ethereum IBFT2
Hardhat	2.28.6	Smart contracts framework
Solidity	0.8.20	Smart contracts language
Blockscout	6.6.0	Block explorer
Prometheus	v2.51.2	Metrics collection
Grafana	10.4.2	Metrics visualization
MetaMask	v11+	Browser Ethereum wallet

Source: versions experimentally validated during implementation.

A.1 Preparing the Ubuntu 22.04 LTS System

The operating system is prepared with the necessary dependencies, including Node.js 22 LTS (Hardhat 2 requires Node.js 22, not 20), Docker Engine, and the basic tools:

```
# System update and dependency installation
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl wget git build-essential software-properties-common

# Node.js 22 LTS
```

```
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt install -y nodejs
# Verificare: node --version  →  v22.x.x

# Docker Engine
curl -fsSL https://get.docker.com | sudo bash
sudo usermod -aG docker $USER

# Python and Ethereum libraries (for key generation)
sudo apt install -y python3-pip
python3 -m pip install eth-keys eth-account --break-system-packages

# IMPORTANT: Docker log rotation (without this step, disk fills up in hours)
sudo bash -c 'cat > /etc/docker/daemon.json << EOF
{"log-driver":"json-file","log-opts":{"max-size":"10m","max-file":"3"}}
EOF'
```

A.2 Creating the Project Structure

The project directory structure is created:

```
mkdir -p ~/ethereum-identity/blockchain/{bootnode, validator1, validator2,
validator3, validator4, observer1, observer2}
mkdir -p ~/ethereum-
identity/{contracts/{identity,credentials,access},scripts,test,deployments}
mkdir -p ~/ethereum-identity/monitoring/{grafana/{provisioning,dashboards}}
cd ~/ethereum-identity/blockchain
```

A.3 Generating Cryptographic Keys for Nodes

Hyperledger Besu generates secp256k1 key pairs for validator and observer nodes via its CLI. One key pair is generated for each of the 4 validator nodes and 2 observer nodes:

```
# Generate keys for each node (repeat for validator1..4, observer1..2)
besu --data-path=validator1 public-key generate
# → displays: Public key: 0x<PUBKEY_V1>  |  Address: 0x<ADDR_V1>

# Record the 4 validator addresses — required in genesis.json:
# ADDR_V1 = 0x<validator1 address>
# ADDR_V2 = 0x<validator2 address>
# ADDR_V3 = 0x<validator3 address>
# ADDR_V4 = 0x<validator4 address>

# Generate the IBFT2 extraData field (RLP-encoded, per Definition 9.2 in the
paper)
besu rlp encode --from=ibft2validators.json --type=IBFT_EXTRA_DATA
# ibft2validators.json contains the list of the 4 validator addresses
```

A.4 Configuring the Besu IBFT2 Network — genesis.json

The genesis.json file defines the initial parameters of the private network. The key parameters correspond to the values from Chapter 9.2: chainId 1337, blockperiodseconds 2, requesttimeoutseconds 4, epochlength 30000, gasLimit 0x7A1200 (8,000,000). Validator addresses are included in the extraData field in RLP format:

```
{
```

```
  "config": {
    "chainId": 1337,
    "berlinBlock": 0,
    "ibft2": {
      "blockperiodseconds": 2,
      "epochlength": 30000,
      "requesttimeoutseconds": 4
    }
  },
  "nonce": "0x0",
  "timestamp": "0x0",
  "gasLimit": "0x7A1200",
  "difficulty": "0x1",
  "extraData": "<IBFT2_RLP_ENCODED_VALIDATORS>",
  "alloc": {
    "<ADDR_V1>": { "balance": "1000000000000000000000" },
    "<ADDR_V2>": { "balance": "1000000000000000000000" },
    "<ADDR_V3>": { "balance": "1000000000000000000000" },
    "<ADDR_V4>": { "balance": "1000000000000000000000" }
  }
}
```

A.5 Configuring Docker Compose for the Blockchain Network

The network is orchestrated via Docker Compose. The service structure reflects the architecture from Figure 9.1: four validator nodes (besu-validator1..4) participating in IBFT2 consensus and two observer nodes (besu-observer1..2) with read-only access via JSON-RPC. Validator 1 exposes the RPC/WebSocket ports for Hardhat and Blockscout:

```
services:
  besu-validator1:
    image: hyperledger/besu:24.x
    container_name: besu-validator1
    volumes:
      - ./genesis.json:/config/genesis.json
      - ./validator1:/data
    command: >
      --data-path=/data
      --genesis-file=/config/genesis.json
      --rpc-http-enabled --rpc-http-host=0.0.0.0 --rpc-http-port=8545
      --rpc-http-api=ETH,NET,IBFT,TXPOOL,DEBUG
      --rpc-ws-enabled --rpc-ws-host=0.0.0.0 --rpc-ws-port=8546
      --p2p-port=30303
      --metrics-enabled --metrics-host=0.0.0.0 --metrics-port=9545
      --bootnodes=enode://<PUBKEY_V2>@besu-validator2:30303,...
    ports: ["8545:8545","8546:8546","9545:9545","30303:30303"]
    networks: [eth-net]

# besu-validator2, besu-validator3, besu-validator4 — similar configuration
# besu-observer1, besu-observer2 — without --miner-enabled, RPC read-only only

networks:
  eth-net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.25.0.0/24
```

Network initialization and startup:

```
# Blockchain initialization (one-time only)
docker compose up -d

# Verify block production (must increase with each call)
curl -s -X POST http://localhost:8545 \
  -H "Content-Type: application/json" \
  -d '{"jsonrpc":"2.0","method":"eth_blockNumber","params":[],"id":1}'

# Verify active IBFT2 validators
curl -s -X POST http://localhost:8545 \
  -d '{"jsonrpc":"2.0", "method":"ibft_getValidatorsByBlockNumber",
"params":["latest"], "id":1}'
# → must return the list of 4 validator addresses
```

A.6 Configuring the Monitoring Stack (Prometheus + Grafana)

The monitoring stack collects metrics from all 4 validator nodes via the Prometheus endpoints exposed on port 9545:

```
# prometheus.yml — collect metrics from all 4 validators
scrape_configs:
  - job_name: besu-validator1
    static_configs:
      - targets: ["besu-validator1:9545"]
    metrics_path: /metrics
# Repeat for validator2, validator3, validator4

# Start monitoring stack
docker compose -f docker-compose.monitoring.yml up -d

# Verify Prometheus targets
curl -s http://localhost:9090/api/v1/targets | python3 -m json.tool | grep
health
# → all 4 jobs must be "health": "up"
```

Grafana dashboards are accessible at http://localhost:3000 (admin/admin1234) and display: number of blocks produced, TPS rate, confirmation latency, number of connected peers, and resource consumption per node.

A.7 Configuring the Blockscout Block Explorer

Blockscout 6.6.0 is configured to index the private network by connecting to validator1 via JSON-RPC and WebSocket. PostgreSQL 15 and Redis 7 are required as backend dependencies:

```
# Required environment variables for Blockscout
ETHEREUM_JSONRPC_VARIANT=geth
ETHEREUM_JSONRPC_HTTP_URL=http://besu-validator1:8545
ETHEREUM_JSONRPC_WS_URL=ws://besu-validator1:8546
NETWORK=Private ETH Identity Chain
SUBNETWORK=IBFT2 PoA (Besu)
CHAIN_ID=1337

# Run migrations (mandatory before first startup)
docker run --rm --network eth-net \
  -e DATABASE_URL=postgresql://blockscout:pass@blockscout-db:5432/blockscout \
  blockscout/blockscout:6.6.0 \
```

```
  /bin/sh -c "/app/bin/blockscout eval 'Explorer.ReleaseTasks.migrate()'"

# Start Blockscout backend + frontend
docker compose -f docker-compose.explorer.yml up -d
# UI available at: http://localhost:3001
```

A.8 Configuring the Hardhat Project and Deploying Contracts

The Hardhat 2 project manages the compilation and deployment of the three smart contracts on the private network:

```
# Initialize Hardhat 2 project
cd ~/ethereum-identity
npm init -y
npm install --save-dev hardhat@2 @openzeppelin/contracts dotenv
npm install --save-dev "@nomicfoundation/hardhat-toolbox@hh2"
npx hardhat  # select "A Javascript project using Mocha and Ethers.js"

# hardhat.config.js — private Besu network configuration
networks: {
  privateChain: {
    url: "http://localhost:8545",
    chainId: 1337,
    accounts: [process.env.DEPLOYER_PRIVATE_KEY],
    timeout: 60000
  }
}
solidity: { version: "0.8.20", settings: { optimizer: { enabled: true, runs: 200
} } }

# Compile contracts
npx hardhat compile
# → Compiled 3 Solidity files successfully (evm target: paris)

# Deploy all 3 contracts to the private network
npx hardhat run scripts/deploy.js --network privateChain
# → EthereumDIDRegistry deployed: 0x...
# → VCStatusRegistry deployed: 0x...
# → AccessControlRegistry deployed: 0x...
# → Addresses saved to deployments/addresses.json
```

A.9 Configuring MetaMask

MetaMask is used for manual interaction with the private network and balance verification. The private network is added with the parameters from Table A.2:

Table A.2. Private network parameters in MetaMask.

Field	Value
Network name	Private ETH Identity Chain
Default RPC URL	http://localhost:8545
Chain ID	1337
Currency symbol	ETH

Block explorer URL	http://localhost:3001

A.10 Final Environment Verification Checklist

Successful completion of all checks confirms that the working environment is fully operational:

Table A.3. Experimental environment verification checklist.

#	Check	Command / URL
1	Blockchain: blocks produced at ~2 seconds	curl ... eth_blockNumber
2	IBFT2: 4 active validators	curl ... ibft_getValidatorsByBlockNumber
3	Prometheus: all 4 targets health: up	http://localhost:9090/targets
4	Grafana: interface accessible	http://localhost:3000
5	Blockscout: total_blocks > 0 and increasing	http://localhost:4000/api/v2/stats
6	Blockscout UI: blocks visible	http://localhost:3001
7	Hardhat: compilation successful	npx hardhat compile
8	Deploy: all 3 contracts visible in Blockscout	curl .../api/v2/addresses/...
9	MetaMask: private network selected, balance ~1000 ETH	Visual browser verification

Source: own elaboration.

A.11 Quick Reference for Ports and Services

Table A.4. Port and access URL reference for the experimental environment.

Service	Port	URL	Purpose
Besu Validator1 RPC	8545	http://localhost:8545	API JSON-RPC Hardhat
Besu Validator1 WS	8546	ws://localhost:8546	Event subscriptions
Besu Metrics	9545	intern Docker	Metrici Prometheus
Prometheus	9090	http://localhost:9090	UI Prometheus
Grafana	3000	http://localhost:3000	Dashboards
Blockscout API	4000	http://localhost:4000	REST API
Blockscout UI	3001	http://localhost:3001	Web explorer interface

Source: own elaboration.

A.12 Smart Contract Pseudocode

This section presents the abstract algorithmic logic of the three implemented contracts, using a language-independent pseudocode notation inspired by Solidity.

Pseudocode A.1 — EthereumDIDRegistry.sol (Decentralized Identity Registry)

```
CONTRACT EthereumDIDRegistry:

  STORAGE:
    owners    : mapping(address → address)
    changed   : mapping(address → uint256)   // ultimul bloc modificat
    delegates : mapping(address → mapping(bytes32 → mapping(address → uint256)))

  FUNCTION setAttribute(identity, name, value, validity):
    REQUIRE ownerOf(identity) == msg.sender
    validTo ← block.timestamp + validity
    changed[identity] ← block.number
    EMIT DIDAttributeChanged(identity, name, value, validTo, previousChange)

  FUNCTION revokeAttribute(identity, name, value):
    REQUIRE ownerOf(identity) == msg.sender
    changed[identity] ← block.number
    EMIT DIDAttributeChanged(identity, name, value, validTo=0, previousChange)

  FUNCTION changeOwner(identity, newOwner):
    REQUIRE ownerOf(identity) == msg.sender
    owners[identity] ← newOwner
    changed[identity] ← block.number
    EMIT DIDOwnerChanged(identity, newOwner, previousChange)

  FUNCTION ownerOf(identity) → address:
    IF owners[identity] == 0x0 THEN RETURN identity
    ELSE RETURN owners[identity]
```

Pseudocode A.1. Logic of the EthereumDIDRegistry.sol contract. [Original contribution]

Pseudocode A.2 — VCStatusRegistry.sol (Verifiable Credential Status Registry)

```
CONTRACT VCStatusRegistry:

  ENUM StatusType: UNDEFINED | ISSUED | REVOKED | SUSPENDED

  STORAGE:
    credentials   : mapping(bytes32 → CredentialStatus)
// CredentialStatus = { status, issuedAt, issuer, evidenceHash }
    statusListHash : mapping(uint256 → bytes32)  // index → IPFS CID hash

  FUNCTION issueCredential(vcId, evidenceHash):
    REQUIRE hasRole(ISSUER_ROLE, msg.sender)
    REQUIRE credentials[vcId].status == UNDEFINED
    credentials[vcId] ← { ISSUED, block.timestamp, msg.sender, evidenceHash }
    EMIT CredentialIssued(vcId, msg.sender, block.timestamp)

  FUNCTION revokeCredential(vcId):
    REQUIRE hasRole(ISSUER_ROLE, msg.sender)
    REQUIRE credentials[vcId].issuer == msg.sender
    REQUIRE credentials[vcId].status == ISSUED
    credentials[vcId].status ← REVOKED
    EMIT CredentialRevoked(vcId, msg.sender, block.timestamp)

  FUNCTION suspendCredential(vcId):
    REQUIRE hasRole(ISSUER_ROLE, msg.sender)
    credentials[vcId].status ← SUSPENDED
    EMIT CredentialSuspended(vcId, msg.sender, block.timestamp)

  FUNCTION getStatus(vcId) → StatusType:
    RETURN credentials[vcId].status
```

Pseudocode A.2. Logic of the VCStatusRegistry.sol contract. [Original contribution]

Pseudocode A.3 — AccessControlRegistry.sol (Role-Based Access Control)

```
CONTRACT AccessControlRegistry EXTENDS OpenZeppelin.AccessControl:

  ROLES:
    ADMIN_ROLE    ← DEFAULT_ADMIN_ROLE
    ISSUER_ROLE   ← keccak256("ISSUER_ROLE")
    VERIFIER_ROLE ← keccak256("VERIFIER_ROLE")
    AUDITOR_ROLE  ← keccak256("AUDITOR_ROLE")

  CONSTRUCTOR(initialAdmin):
    grantRole(ADMIN_ROLE, initialAdmin)
    setRoleAdmin(ISSUER_ROLE,   ADMIN_ROLE)
    setRoleAdmin(VERIFIER_ROLE, ADMIN_ROLE)
    setRoleAdmin(AUDITOR_ROLE,  ADMIN_ROLE)

  FUNCTION grantIssuerRole(account):
    REQUIRE hasRole(ADMIN_ROLE, msg.sender)
    grantRole(ISSUER_ROLE, account)
    EMIT IssuerGranted(account, msg.sender)

  FUNCTION revokeIssuerRole(account):
    REQUIRE hasRole(ADMIN_ROLE, msg.sender)
    revokeRole(ISSUER_ROLE, account)
    EMIT IssuerRevoked(account, msg.sender)

  FUNCTION checkAccess(account, role) → bool:
    RETURN hasRole(role, account)

// Modifier reused in VCStatusRegistry and EthDIDRegistry:
  MODIFIER onlyRole(role):
    REQUIRE hasRole(role, msg.sender)
    EXECUTE function body
```

Pseudocode A.3. Logic of the AccessControlRegistry.sol contract. [Original contribution]

References

1. Antonopoulos, A. M., & Wood, G. (2018). Mastering Ethereum: Building smart contracts and DApps. O'Reilly Media.
2. Back, A. (1997). Hashcash — A denial of service counter-measure. http://www.hashcash.org/papers/hashcash.pdf
3. Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. IACR ePrint 2018/046.
4. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., & Virza, M. (2014). Succinct non-interactive arguments for a von Neumann architecture. USENIX Security Symposium, 781–796.
5. Boneh, D., Boyen, X., & Shacham, H. (2004). Short group signatures. In Advances in Cryptology — CRYPTO 2004, Lecture Notes in Computer Science, 3152, 41–55.
6. Brewer, E. A. (2000). Towards robust distributed systems. Proceedings of 19th ACM PODC, 7.
7. Buterin, V. (2014). Ethereum: A next-generation smart contract and decentralized application platform. Ethereum Foundation. https://ethereum.org/en/whitepaper/
8. Camenisch, J., & Lysyanskaya, A. (2004). Signature schemes and anonymous credentials from bilinear maps. In Advances in Cryptology — CRYPTO 2004, Lecture Notes in Computer Science, 3152, 56–72.
9. Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. USENIX OSDI, 173–186.
10. Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2), 374–382.
11. Gabizon, A., Williamson, Z. J., & Ciobanu, O. (2019). PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR ePrint 2019/953.

12. Groth, J. (2016). On the size of pairing-based non-interactive arguments. In Advances in Cryptology — EUROCRYPT 2016, Lecture Notes in Computer Science, 9666, 305–326.
13. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., & Gervais, A. (2020). SoK: Layer-two blockchain protocols. In Financial Cryptography 2020, Lecture Notes in Computer Science, 12059.
14. Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. Journal of Cryptology, 3(2), 99–111.
15. Hyperledger Besu Documentation. (2024). IBFT 2.0 proof of authority. https://besu.hyperledger.org/stable/private-networks/how-to/configure/consensus/ibft
16. Hyperledger Foundation. (2023). Hyperledger Fabric documentation v2.5. Linux Foundation. https://hyperledger-fabric.readthedocs.io
17. Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security, 1, 36–63.
18. Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4(3), 382–401.
19. Merkle, R. C. (1980). Protocols for public key cryptosystems. IEEE Symposium on Security and Privacy, 122–134.
20. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf
21. NIST. (2024). Post-quantum cryptography standards (FIPS 203, 204, 205). National Institute of Standards and Technology. https://csrc.nist.gov/pqcrypto
22. OpenZeppelin. (2024). OpenZeppelin contracts v5.0. https://docs.openzeppelin.com/contracts
23. Schnorr, C. P. (1991). Efficient signature generation by smart cards. Journal of Cryptology, 4(3), 161–174.
24. Sporny, M., Longley, D., & Chadwick, D. (2022). Verifiable credential status list 2021. W3C Community Group Final Report.
25. Szabo, N. (1997). Formalizing and securing relationships on public networks. First Monday, 2(9).
26. W3C. (2022a). Decentralized identifiers (DIDs) v1.0. W3C Recommendation, 19 July 2022. https://www.w3.org/TR/did-core/
27. W3C. (2022b). Verifiable credentials data model v1.1. W3C Recommendation, 3 March 2022. https://www.w3.org/TR/vc-data-model/
28. Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger (Yellow Paper). Ethereum Foundation. https://ethereum.github.io/yellowpaper/paper.pdf
29. Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An overview of blockchain technology: Architecture, consensus, and future trends. IEEE BigData Congress, 557–564.