Sneaky Blockchain

Technical White Paper

For Developers

22/11/2024

Contents

This Paper..2

1. Introduction..2

2. Technology Stack..4

3. Core Architecture..10

4. Services Layer and Technical Implementation..............22

5. Installation and Setup...................................25

6. Block Handling and Propagation...........................31

6.   Security and Encryption 39

8. Network Reliability and Worker Management................44

9. AI and Automation in Sneaky..............................48

10. Open Source and Community Engagement....................51

11. Obvious Best Practice...................................54

12. Conclusion..61

Get Started...63

Bibliography..64

This Paper

This is the second of two Sneaky white papers. The first covered business aspects for managers. This technical paper targets developers. We will expand it as our code evolves.

1. Introduction

1.1 Overview

Sneaky is a high-performance permissioned blockchain. It brings three key improvements to distributed ledger technology:

- Enhanced efficiency
- Better scalability
- Minimal energy use

Sneaky differs from traditional blockchains. It eliminates decentralized consensus. Instead, it uses a central clock node. This clock orders and signs all transactions.

Security comes from cryptographic proofs rather than consensus. This design creates two advantages:

- Faster data processing
- Lower resource consumption

These features make Sneaky ideal for businesses needing fast, cost-effective blockchain solutions.

1.2 Objectives and Vision

Sneaky prioritizes three core elements:

- Technical accuracy
- System reliability
- Industry best practices

Every system component undergoes rigorous testing. The platform handles heavy workloads through:

- Clear workflows
- Strong encryption
- Stable blockchain environment

This approach delivers:

- High security
- Consistent performance
- Easy business integration
- Future-ready scalability

From an acquisition standpoint, Sneaky follows clear strategic principles:

- Adherence to technical standards
- Transparent development practices
- Enterprise-ready architecture
- Seamless tech firm integration

1.3 Core Principles

Technical Correctness and Verification

Sneaky maintains strict technical accuracy. The system includes:

- Comprehensive testing protocols
- Verified reliability metrics
- Cryptographic identity checks
- Validated node interactions

This verification ensures:

- System security

- Operational reliability
- Critical application standards

Best Practices and Industry Standards

Sneaky implements recognized standards across all areas:

- Security protocols
- Scaling mechanisms
- Data handling procedures

Key technical features include:

- Elliptic curve cryptography (ECC)
- Optional post-quantum security
- Energy-efficient clock architecture
- Open-source transparency

2. Technology Stack

Sneaky uses modern technologies. These ensure energy efficiency, enterprise-readiness, and cost-effectiveness. Each component boosts speed, security, and development efficiency.

2.1 Core Languages and Frameworks

C# and .NET

- Primary development platform
- Provides stable, high-performance foundation
- Offers robust library support
- Enables modular design
- Suits enterprise requirements

Blazor

- Powers all block explorers

- Enables interactive web interfaces
- Handles node setup workflows
- Manages user registration
- Monitors transactions
- Uses server-side rendering for speed

2.2 Storage and Data Management

SQLite

- Manages local node storage
- Handles transaction pools
- Maintains block pools
- Stores chain data
- Provides reliability with simplicity

Azure Blob Storage (Optional)

- Supports cloud deployments
- Ensures data persistence
- Enables large-scale access
- Offers geographic redundancy

2.3 Communication Protocols

Sneaky uses modern, proven protocols for network communication. These choices ensure reliable operation. They maintain system simplicity.

gRPC Communication

Sneaky uses gRPC as its primary communication protocol. Google developed gRPC for high-performance systems. It offers several key advantages.

gRPC uses HTTP/2 for transport. This provides multiplexed connections. It enables bi-directional streaming. It automatically compresses data. These features make communication fast and efficient.

Binary serialization makes messages compact. This reduces network load. It speeds up processing. It lowers bandwidth usage. These benefits help Sneaky maintain high performance.

Strong typing prevents data errors. Auto-generated code reduces bugs. Built-in security features protect data. These characteristics make gRPC both safe and reliable.

HTTPS Security

All external communications use HTTPS. This protects data moving between nodes. It prevents message interception. It ensures data integrity. Standard port 443 ensures wide compatibility.

Fallback Systems

Sneaky includes automatic protocol fallback. If gRPC encounters issues, the system switches to WebSocket. This happens automatically. Users notice no interruption. The system maintains operation even in restricted networks.

Network Resilience

Our protocol choices work in real-world conditions. They handle network interruptions gracefully. They recover from errors automatically. They adapt to different network environments. This makes Sneaky reliable in any setting.

These communication choices support Sneaky's core principles. They favour simplicity over complexity. They choose proven standards over custom protocols. They ensure reliable operation without unnecessary features.

2.4 Security and Cryptography

Elliptic Curve Cryptography (ECC)

- Handles key generation
- Manages digital signatures
- Provides strong security
- Reduces computational overhead

Post-Quantum Encryption (Optional)

- Offers quantum-resistant security
- Future-proofs the system
- Protects against emerging threats
- Maintains backward compatibility

Secure Local Storage

- Isolates credentials
- Protects private keys
- Prevents unauthorized access
- Maintains node security

2.5 AI Integration with SneakyGPT

SneakyGPT makes blockchain data accessible to everyone. It removes technical barriers. It enables natural communication with the blockchain.

Natural Language Interface

Users ask questions in plain language. SneakyGPT understands blockchain concepts. It translates queries into precise data requests. It returns clear, understandable answers. This makes blockchain data accessible to non-technical users.

Real-Time Learning

SneakyGPT updates with each new block. It understands the latest chain state. It learns from new transactions. It adapts to system changes. This ensures current, accurate responses.

Secure Operation

SneakyGPT never handles private keys. It cannot modify blockchain data. It only reads verified information. It respects all security boundaries. These limits maintain system security.

Business Intelligence

The system recognizes business patterns. It identifies transaction trends. It spots unusual activities. It generates custom reports. This helps organizations understand their blockchain usage.

Query Processing

SneakyGPT converts natural language to precise queries. It checks multiple data points. It combines relevant information. It formats responses clearly. Users receive exactly what they need.

Practical Benefits

Business users avoid complex query languages. Managers get quick insights. Auditors find relevant transactions easily. Developers save time. These benefits speed up blockchain adoption.

SneakyGPT makes blockchain practical for everyday use. It maintains security while improving access. It helps organizations embrace blockchain technology. It turns complex data into useful information.

Future Developments

Future Developments

SneakyGPT is at the start of its development curve. It is the simplest possible implementation.

Voice interaction comes first. Users will speak to their blockchain. The blockchain will speak back. This makes blockchain easier to use.

Pattern recognition will get better. SneakyGPT will find trends by itself. It will suggest useful questions. It will point to important data. Your blockchain data becomes more useful.

Security watching will grow stronger. SneakyGPT will learn what normal looks like. It will spot unusual events quickly. It will help stop problems early. Your system stays safer.

Report creation will improve. Users will get personalized analysis. They will receive smarter suggestions. They will see better comparisons. Your business gains more value.

Our core principle stays the same. New features must keep things simple. Every addition must have a purpose. All changes must protect security. SneakyGPT will grow without complexity.

2.6 Clock Redundancy

Simple, Effective Approach

Sneaky uses a straightforward approach to clock reliability. In our SneakyB public blockchain[1]:

[1] www.sneakyb.au

- Clock runs on Azure cloud services
- Benefits from standard Azure redundancy
- Maintains simplicity of design
- Avoids complex infrastructure

Enterprise Implementation

For enterprise deployments, clock reliability follows the same principle of simplicity:

1. Host on your preferred cloud platform
2. Use built-in cloud redundancy features
3. Back up clock node private keys
4. Keep standard monitoring

Why This Works

Sneaky's design principles focus on:

- Simplicity over complexity
- Standard cloud features over custom solutions
- Reliable basic operations over complex redundancy
- Easy maintenance over elaborate systems

This approach keeps Sneaky:

- Light and efficient
- Easy to implement
- Simple to maintain
- Cost-effective
- Reliably functional

3. Core Architecture

3.1 Key Components

Sneaky's architecture centres on three main components. Each serves specific functions for reliability, efficiency, and scalability.

Clock Node

The clock node is SneakyB’s innovative central mechanism for low-energy transaction sequencing and verification. This centralized clock allows SneakyB to process transactions quickly without energy-heavy consensus. It provides an enterprise-ready solution that lowers costs and simplifies infrastructure.

Primary Functions:

- Acts as central control point
- Sets transaction sequence
- Applies cryptographic signatures
- Verifies incoming blocks
- Ensures worker synchronization
- Eliminates consensus energy waste

Worker Nodes

Worker nodes verify transactions and create blocks in a cost-effective, streamlined way. This setup makes SneakyB scalable and reliable for enterprise workloads. It also uses less energy than traditional blockchains.

Primary Functions:

- Connect senders to clock
- Collect sender transactions
- Verify transaction validity
- Package transactions into blocks
- Submit blocks for clock signing
- Maintain chain copies

- Share blocks across network
- Ensure network consistency

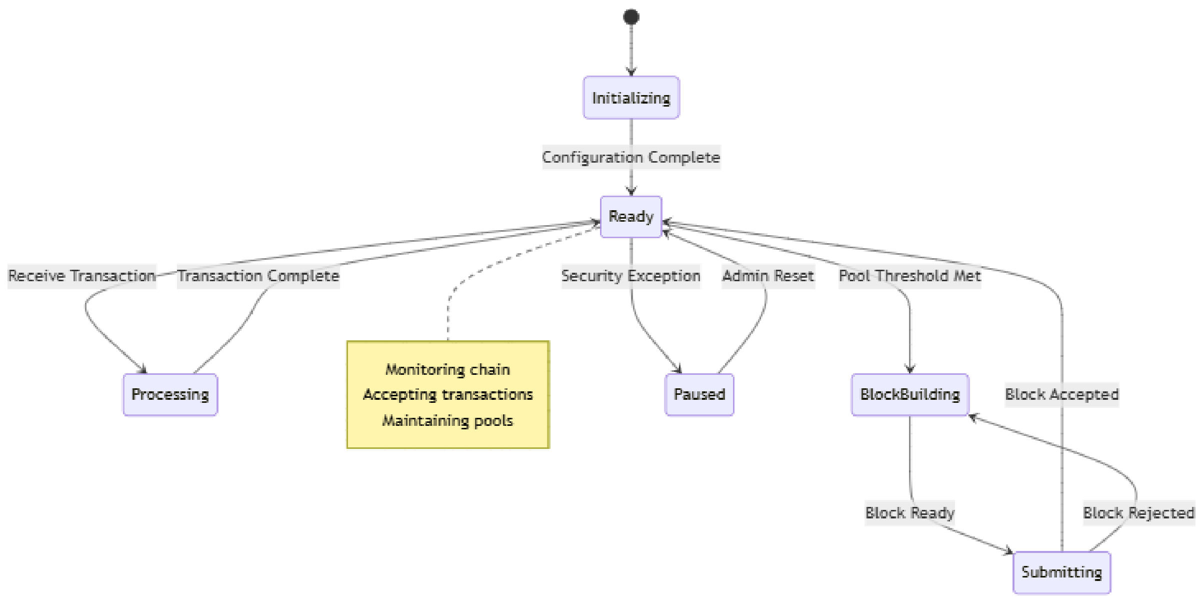

Figure 1: Worker node state transitions showing operational states, processing conditions, and error handling.

Sender Nodes

Sender nodes generate and submit transactions with minimal technical demands for enterprises. Any device that can reach a worker connection can act as a sender node. Sneaky service calls integrate into any programming language. Sender nodes also keep local chain copies to allow cost-efficient access and save bandwidth. This lightweight design supports SneakyB's goal of scalable, low-cost data integrity for enterprises.

Core Operations:

- Generate transactions

- Connect to worker nodes
- Send data to chain
- Store local chain copies (shards)
- Maintain transaction footprints
- Enable local data access

3.2 Data Classes and Structures

Sneaky uses four core data classes to handle blockchain operations and message authentication.

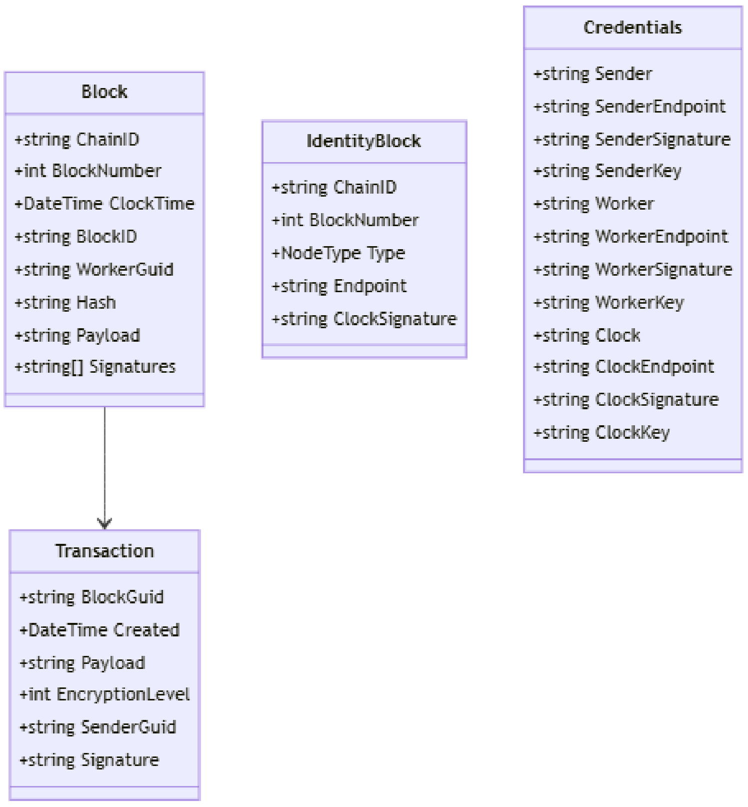

Figure 2: Core data structure relationships. Shows how Block, Transaction, IdentityBlock, and Credentials classes interact within the system.

Block Class

The Block class forms the fundamental structure of the blockchain. Each block contains a collection of verified transactions, uniquely identified through its BlockID. The block maintains its position and authenticity through a combination of chain ID and block number assignments. A timestamp from the clock node ensures precise temporal ordering, while worker information and digital signatures verify the block's origin and validity. The block's payload contains the actual transaction data, protected by comprehensive hash values that ensure data integrity.

Key Components:

- Unique identifier (BlockID)
- Block number and chain ID
- Clock timestamp
- Worker information
- Hash values
- Digital signatures
- Transaction payload

C# Implementation:

```
public class Block
{
    public string ChainID { get; set; }
    public int BlockNumber { get; set; }
    public DateTime ClockTime { get; set; }
    public string BlockID { get; set; }
    public string WorkerGuid { get; set; }
    public string Hash { get; set; }
    public string SHash { get; set; }
    public string Payload { get; set; }
    public string ClockSignature { get; set; }
    public string WorkerSignature { get; set; }
}
```

Transaction Class

Transactions represent individual operations within the blockchain system. Each transaction carries its own unique identifiers and timestamps, establishing its place within the block structure. The transaction payload contains the actual business data being stored, with encryption settings determining data protection levels. Digital signatures verify the transaction's origin, while file handling capabilities enable diverse data storage needs. This flexible structure allows Sneaky to accommodate various business use cases while maintaining security and traceability.

Key Components:

- Transaction identifiers
- Creation timestamp
- Data payload
- Encryption settings
- File handling data
- Digital signatures

C# Implementation:

```
public class Transaction
{
    public string BlockGuid { get; set; }
    public string Guid { get; set; }
    public DateTime Created { get; set; }
    public string Payload { get; set; }
    public int EncryptionLevel { get; set; }
    public string ChainID { get; set; }
    public string SenderGuid { get; set; }
    public string WorkerGuid { get; set; }
    public string SenderSignature { get; set; }
    public bool IsFile { get; set; }
    public string Filename { get; set; }
    public string OriginalFilename { get; set; }
}
```

Credentials Class

Unlike the other classes, the Credentials class operates outside the blockchain storage system, focusing instead on message authentication between nodes. It plays a crucial role in securing inter-node communications through a comprehensive set of authentication components. When nodes communicate, each message includes a serialized Credentials object that contains node identifiers, network endpoints, public keys, and role-specific digital signatures.

The Credentials class authenticates three primary message types: Transaction Push messages from senders to workers, Block Push messages from workers to the clock, and Block Share messages between workers. For each message, the Credentials object includes the necessary sender, worker, or clock credentials required for that specific interaction. This authentication system ensures that only authorized nodes can participate in blockchain operations, maintaining the security of the entire system.

Message Types:

- Transaction Push (Sender → Worker)
- Block Push (Worker → Clock)
- Block Share (Worker → Worker)

Authentication Components:

- Node identifiers
- Network endpoints
- Public keys
- Digital signatures for each node type:
    - Sender credentials
    - Worker credentials
    - Clock credentials

Each message between nodes includes a serialized Credentials object for authentication.

C# Implementation:

```
public class Credentials
{
    public string Sender { get; set; }
    public string SenderEndpoint { get; set; }
    public string SenderSignature { get; set; }
    public string SenderKey { get; set; }
    public string Worker { get; set; }
    public string WorkerEndpoint { get; set; }
    public string WorkerSignature { get; set; }
    public string WorkerKey { get; set; }
    public string Clock { get; set; }
    public string ClockEndpoint { get; set; }
    public string ClockSignature { get; set; }
    public string ClockKey { get; set; }
}
```

IdentityBlock Class

Functions:

Identity blocks serve as the blockchain's trust anchors, recording and verifying the identity of each participating node. When a node joins the network, an identity block captures its role, network location, and verification data. This information becomes part of the blockchain itself, creating an immutable record of node participation. The clock node's signature on each identity block ensures the authenticity of node registration, maintaining the integrity of the network's trust structure.

Key Components:

- Node identification
- Role information
- Network location
- Verification data
- Chain position

C# Implementation:

```
public class IdentityBlock
{
    public string ChainID { get; set; }
    public int BlockNumber { get; set; }
    public DateTime ClockTime { get; set; }
    public string BlockID { get; set; }
    public string Hash { get; set; }
    public int Type { get; set; }
    public string Endpoint { get; set; }
    public string Payload { get; set; }
    public string ClockSignature { get; set; }
    public enum NodeType { Unknown, Sender, Worker }
}
```

System Benefits:

- Clear component roles
- Efficient data handling
- Strong security model
- Simple management
- Easy verification

This architecture provides:

- Focused component responsibilities
- Clear data flow
- Strong security
- Easy maintenance
- High performance

3.3 Settings Management

Each Sneaky node maintains its own settings configuration. This includes:

- Local preferences
- Parent node settings

- Network parameters

Configuration System

Sneaky uses .NET's configuration framework:

- Stores settings in appsettings.json
- Supports multiple environments
- Enables quick updates
- Maintains persistence
- Allows easy backup

Storage Implementation

File Structure:

- Primary configuration file
- Environment-specific versions
- Local overrides
- Backup copies
- Update logs

Configuration Types

Core Settings:

- API endpoints
- Block parameters
- Timeout values
- Security configs
- Network details

Management Interface

Block Explorer Features:

- Settings dashboard

- Visual controls
- Real-time updates
- Value validation
- Change tracking

Settings Hierarchy

Update Flow:

1. Clock sets master config
2. Workers receive updates
3. Senders get worker settings
4. Local values merge
5. System syncs changes

Environment Support

Deployment Options:

- Development settings
- Production configs
- Testing parameters
- Staging values
- Custom environments

Dynamic Updates

System Features:

- Automatic reload
- Live configuration
- No restart needed
- Instant propagation
- Version tracking

Technical Benefits:

- Reliable storage
- Easy management
- Quick updates
- Clear hierarchy
- Strong consistency

System Advantages:

- Simple configuration
- Flexible control
- Fast deployment
- Easy maintenance
- Network synchronization

3.4 Worker Node Operations

Worker nodes form the backbone of Sneaky's transaction processing. Each worker manages a critical set of operations. These operations keep the blockchain running smoothly.

Transaction Management

Workers receive transactions from sender nodes. They verify each transaction's authenticity. They check digital signatures against the blockchain record. Valid transactions enter the worker's transaction pool. Invalid transactions trigger security alerts.

Block Building

Workers monitor their transaction pools. When enough transactions accumulate, the worker builds a new block. It arranges transactions in chronological order. It adds its own signature to the block. It then submits this block to the clock node for verification.

Network Communication

Workers maintain connections with multiple system components. They listen for new transactions from senders. They submit blocks to the clock node. They share verified blocks with other workers. This communication pattern keeps the network synchronized without complex consensus mechanisms.

Chain Maintenance

Each worker keeps a copy of the blockchain. It adds new blocks as they are verified. It shares blocks with workers that rejoin the network. It maintains transaction history for its senders. This local storage enables quick verification and response.

State Management

Workers track their own operational state. They monitor their connection to the clock. They manage their transaction pools. They track their block building status. This self-monitoring ensures reliable operations.

Worker nodes simplify blockchain operations. They eliminate the need for network-wide consensus. They provide clear transaction and block handling. They maintain system consistency through simple, proven methods.

4. Services Layer and Technical Implementation

The SneakyB services layer is designed to ensure secure, efficient, and scalable transaction handling. Each node type—Clock, Worker, and Sender—operates with distinct functions and interactions, structured to support SneakyB’s focus on energy efficiency, enterprise-readiness, and cost-effectiveness. SneakyB service calls are designed to integrate with any programming language, maximizing accessibility across different enterprise environments.

Clock Service

Role: The Clock Service is SneakyB’s authoritative timekeeper and central node for transaction sequencing. It ensures a consistent transaction order across the network by timestamping and signing each block.

Functions:

- Receive Blocks: Accepts completed blocks from Worker nodes for sequencing.
- Validate Blocks: Checks each block for integrity and compliance with blockchain rules before it is added to the sequence.
- Timestamp and Sign Blocks: Adds a precise timestamp and digital signature to each block, ensuring its place in the transaction sequence.
- Return Signed Blocks: Sends the signed and sequenced blocks back to Worker nodes to ensure consistency across the network.
- Register Workers: Registers Worker nodes and issues an IdentityBlock to confirm their identity and integration with the network.

C# Implementation[2]:

```
public interface IClockService
{
    Task<Block> PushBlockAsync(Block block);
```

[2] This is a basic implementation. There are more methods in our production classes.

```
    Task<IdentityBlock> RegisterWorkerAsync (Credentials
    credentials);
    Task<List<Block>> WorkerHandshakeAsync (Credentials
    credentials);
    Task<Instruction> Exception (Error error);
}
```

Worker Service

Role: The Worker Service acts as an intermediary between sender nodes and the Clock Service. It manages transaction validation, block formation, and network consistency.

Functions:

- Receive Transactions: Accepts transactions from sender nodes, validating each one before adding it to the pool of transactions.
- Assemble Blocks: Batches validated transactions into blocks, optimizing network resources by reducing the frequency of communication with the Clock node.
- Sync with Peers: Shares completed blocks with other Worker nodes to maintain a consistent chain state across the network, ensuring redundancy and alignment.
- Register Senders: Registers sender nodes, issuing an IdentityBlock for each new sender to establish an initial point of reference within the chain.

C# Implementation[3]:

```
public interface IWorkerService
{
```

[3] This is a basic implementation. There are more methods in our production classes.

```
    Task<bool> PushTransactionAsync (Transaction transaction);
    Task SyncBlockAsync (Block block);
    Task<IdentityBlock> RegisterSenderAsync (Credentials
    credentials);
    Task<List<Block>> SenderHandshakeAsync (Credentials
    credentials);
    Task<Instruction> Exception (Error error);
}
```

Sender Nodes

Currently, there is no sender service. Instead, senders initiate transactions and receive updates in reply to worker method calls and in their worker node handshake at startup.

Summary

This services layer design is optimized for SneakyB’s goals of simplicity, low-cost operation, and security. Each service operates within a limited role to maintain an efficient, secure, and enterprise-ready infrastructure that minimizes resource use and supports scalability across diverse environments.

5. Installation and Setup

5.1 Node Installation

Each node type requires specific installation procedures. Custom installers handle environment setup, configuration, and node interaction protocols.

Clock Node Installation

Setup Process:

- Creates local blockchain instance
- Initializes system parameters
- Generates unique keypair

- Stores private key securely
- Embeds public key in genesis block
- Prepares network management systems

Post-Installation State:

- Ready for network management
- Prepared for block verification
- Enabled for system monitoring

Worker Node Installation

Components:

- Deploys block explorer
- Prepares clock registration
- Generates node keypair
- Sets up security protocols
- Configures transaction handling

Ready State:

- Prepared for clock registration
- Configured for transaction processing
- Enabled for block management

Sender Node Installation

Requirements:

- Sets up worker registration
- Configures gateway access
- Prepares transaction systems
- Establishes security protocols

Registration Process:

- First worker contact triggers registration
- System validates credentials
- Node joins transaction network

5.2 Clock Setup

Clock Configuration Steps:

1. Access local block explorer
2. Sign in with default credentials
3. Complete initial setup wizard
4. Enable worker token generation
5. Configure security parameters

Operational State:

- Ready for worker registration
- Enabled for token generation
- Prepared for block signing
- Configured for network management

5.3 Worker and Sender Registration

Worker Registration Process

Steps:

1. Access clock block explorer
2. Submit worker access token
3. Complete verification process
4. Receive identity confirmation
5. Join blockchain network

System Actions:

- Clock verifies token
- Creates worker identity block

- Enables block submission
- Grants chain interaction rights

Sender Registration Process

Registration Methods:

- API-based registration
- Block explorer token submission

System Flow:

1. Sender submits credentials
2. Worker verifies access rights
3. Worker shares identity with clock
4. Clock creates sender identity block
5. Sender gains transaction rights

Post-Registration State:

- Ready for transaction submission
- Enabled for chain interaction
- Configured for data transfer
- Prepared for shard maintenance

Key Benefits:

- Secure node onboarding
- Clear registration paths
- Strong identity verification
- Simple setup process

5.4 Payload Agnosticism

Sneaky handles any data format:

- Stores all data as strings

- Accepts any payload format
- Supports serialized JSON
- Handles raw text
- Enables flexible business use

Technical Benefits:

- Format independence
- Universal compatibility
- Simple data validation
- Easy content parsing
- Flexible implementation

5.5 Transaction Processing Flow

Sneaky processes transactions through a series of clear, verifiable steps. Each step ensures both security and efficiency.

Initial Transaction Creation

A sender node begins each transaction. It creates a Transaction object containing the business data payload. The sender signs this object with its private key. This signature proves the transaction's origin. The sender then packages this transaction with its credentials for worker verification.

Worker Node Processing

The worker node serves as the transaction's first verification point. It checks the sender's signature against the sender's public key from the blockchain. This confirms the sender's identity. The worker then validates the transaction format and data structure.

Valid transactions enter the worker's transaction pool. Here they wait for block creation. Invalid transactions trigger security alerts. The worker node monitors its pool size against the system's block length setting.

Block Creation

When enough transactions fill the pool, the worker creates a new block. The worker arranges transactions in chronological order within the block. It adds its own signature to prove block origin. The worker then sends this block to the clock node for final verification.

Clock Node Verification

The clock node provides Sneaky's central verification. It first checks the worker's signature using the worker's public key from the blockchain. This proves the block came from an authorized worker. The clock then examines each transaction's sender signature.

After verification, the clock adds its timestamp and signature. This timestamp establishes the block's permanent position in the chain. The clock's signature seals the block's authenticity. The clock then returns the signed block to the worker.

Chain Integration

The worker adds the signed block to its local chain copy. It shares this block with other worker nodes. This keeps all workers synchronized. The worker then confirms successful processing to the original senders.

Error Handling

Sneaky manages errors without disrupting the chain. Invalid signatures trigger security alerts. Malformed transactions receive rejection notices. Network issues trigger automatic retries. This creates a robust, reliable system.

Each step in this process serves a specific purpose:

- Sender signatures prove data origin
- Worker verification prevents invalid transactions
- Clock signatures establish order
- Multiple checks ensure security
- Clear confirmations maintain reliability

This straightforward approach creates a secure, efficient transaction system. It eliminates the complexity of traditional blockchain consensus. It maintains data integrity through simple, proven methods.

6. Block Handling and Propagation

6.1 Block Creation

Initiation Conditions

Worker nodes start block creation when:

- Transaction pool reaches BlockLength
- Clock settings trigger threshold
- Network conditions allow

Block Assembly Process

Worker Steps:

1. Collects validated transactions
2. Creates JSON transaction collection
3. Builds Block object
4. Signs completed block
5. Sends to clock for mining

Block Components:

- Transaction collection
- Worker signature
- Block metadata
- Creation timestamp
- Previous block reference

Block Pool Management

Worker Actions:

- Stores block in local pool
- Awaits clock confirmation
- Maintains block status
- Tracks pending blocks
- Updates pool state

6.2 Clock Verification and Mining

Verification Process

Clock Steps:

1. Receives block from worker
2. Verifies worker signature
3. Checks transaction signatures
4. Validates block structure
5. Confirms block sequence

Processing Outcomes

Valid Blocks:

- Enter main chain
- Receive clock signature
- Get propagation clearance
- Update chain state

Invalid Blocks:

- Trigger error protocols
- Generate system alerts
- Create error logs
- Update security status

Block Distribution

Clock Actions:

- Returns signed block
- Shares missing blocks
- Updates worker settings
- Distributes worker list
- Maintains synchronization

6.3 Worker Propagation

Block Distribution

Worker Steps:

1. Receives mined block
2. Updates local chain
3. Clears block from pool
4. Shares with other workers
5. Updates block status

Network Management

Propagation Handling:

- Tracks successful shares
- Notes failed distributions
- Manages retry attempts
- Updates worker status

- Maintains network state

Shard Management

For Enabled Systems:

- Updates sender shards
- Tracks shard status
- Maintains footprints
- Ensures data consistency
- Validates updates

System Benefits:

- Clear block flow
- Strong verification
- Reliable propagation
- Consistent updates
- Complete tracking

Key Advantages:

- Efficient processing
- Secure verification
- Reliable distribution
- Easy monitoring
- Strong consistency

6.4 Worker Reachability

Worker nodes require guaranteed network connectivity. Sneaky implements an intelligent connection system with automated protocol selection and monitoring.

Connection Protocol Stack

Primary Layer: gRPC over HTTPS

Default Connection:

- Direct gRPC communication
- TLS 1.3 encryption
- HTTP/2 protocol
- Binary serialization
- Automatic compression

Performance Benefits:

- Minimal latency
- Low bandwidth usage
- Quick serialization
- Efficient streaming
- Small headers

Fallback Layer: WebSocket

Automatic Failover:

- Triggers on gRPC failure
- Uses standard port 443
- Maintains persistent connection
- Enables bidirectional flow
- Supports SSL/TLS

Smart Routing System

Dynamic DNS Management

Implementation:

- Cloud-based DDNS service
- Automatic IP updates

- Sub-minute propagation
- Geographic optimization
- Health monitoring

Benefits:

- Handles IP changes
- Reduces downtime
- Optimizes routing
- Improves reliability
- Simplifies management

Intelligent Proxy Layer

Architecture:

- NGINX reverse proxy
- Automatic SSL handling
- Smart load balancing
- Connection pooling
- Request queueing

Security Features:

- DDoS protection
- Rate limiting
- Request filtering
- Access control
- Traffic monitoring

Connection Optimization

Protocol Selection

Automated Process:

1. Tests direct gRPC
2. Measures connection quality
3. Checks alternative routes
4. Selects optimal protocol
5. Monitors performance

Decision Factors:

- Network latency
- Connection stability
- Bandwidth availability
- Security requirements
- Corporate policies

Performance Monitoring

Continuous Checks:

- Latency measurement
- Throughput tracking
- Error rate monitoring
- Protocol performance
- Connection stability

Response Actions:

- Automatic protocol switching
- Route optimization
- Load redistribution
- Alert generation
- Performance logging

Security Implementation

Connection Security

Required Features:

- Mutual TLS authentication
- Certificate validation
- Protocol encryption
- Traffic inspection
- Threat detection

Audit System

Security Tracking:

- Connection logging
- Protocol changes
- Performance metrics
- Security events
- Access attempts

Enterprise Integration

Network Configuration

Deployment Options:

- Corporate proxy support
- Firewall compatibility
- DMZ deployment
- VPN integration
- Cloud connectivity

Management Interface

Control Features:

- Protocol selection
- Performance metrics

- Security settings
- Route management
- Alert configuration

Key Improvements:

1. Automated protocol selection replaces manual TURN setup
2. Enhanced security layers
3. Performance-based routing
4. Continuous monitoring
5. Enterprise-ready features

System Benefits:

- Reduced complexity
- Better performance
- Stronger security
- Easier management
- Higher reliability

7. Security and Encryption

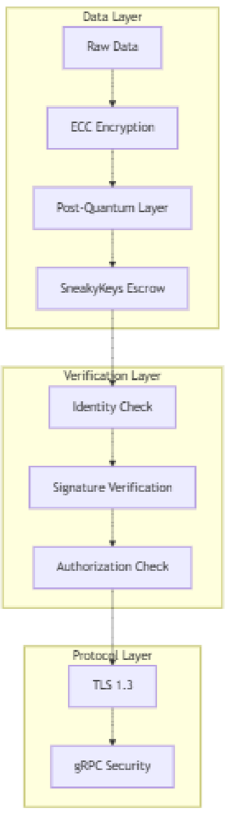

Figure 3: Comprehensive security stack showing encryption layers, verification processes, and protocol security measures.

7.1 ECC and Post-Quantum Readiness

Core Encryption

Sneaky uses Elliptic Curve Cryptography (ECC) as its foundation. ECC provides strong security with small key sizes. Each node creates its own ECC key pair. The private key stays secure on the node. The public key goes into the node's identity block. Every transaction needs a digital signature. Every block needs two signatures. The sender signs each transaction. The worker signs each block. The clock provides the final signature. ECC makes these signatures fast and secure. Small key sizes keep data transfer efficient. Processing overhead stays low. The system maintains security without complexity.

Elliptic Curve Cryptography (ECC):

- Provides primary security layer
- Generates secure keypairs
- Signs all transactions
- Protects node identities
- Ensures data integrity

Technical Benefits:

- Strong security level
- Efficient processing
- Small key sizes
- Fast verification
- Low resource usage

Post-Quantum Features

Optional Protection:

- Resists quantum attacks
- Adds security layer
- Future-proofs data
- Maintains compatibility
- Enables easy upgrades

Implementation Options:

- Full quantum resistance
- Hybrid protection
- Selective encryption
- Gradual migration
- Custom security levels

Post-Quantum Features

Sneaky includes code for future quantum computer protection. This feature is not active in our first release. It will become available in a future update. The feature will be optional for chain owners.

Quantum computers pose an important security challenge. Traditional computers work with bits. Quantum computers use quantum bits or qubits. Qubits can process certain mathematical problems exponentially faster. This includes the math behind current encryption. A powerful quantum computer could break today's encryption methods. It could decode encrypted messages. It could forge digital signatures.

This threat remains theoretical. Current quantum computers are too small. They cannot break encryption yet. Experts estimate this may change within ten years. Post-quantum cryptography prepares for this uncertain future. It includes maths problems that resist quantum attacks. This keeps current data safe even against future quantum computers.

Performance Impact:

- Processing speed drops by 50%
- Storage space doubles
- Memory usage increases by 40%
- Network traffic grows by 60%

Organizations should consider:

- Current security threats
- Performance requirements
- Storage capacity
- Available processing power
- Business security policies

The upgrade process is straightforward for chain owners and transparent to users. The clock node initiates quantum protection. It re-encrypts the existing chain block by block. Workers maintain both chains during transition. Normal service continues throughout. The process typically takes one hour per 10,000 blocks.

We recommend organizations evaluate their needs carefully. Current encryption remains secure against today's threats. Post-quantum features can wait if performance matters more. Each chain owner makes this choice based on their requirements.

7.2 Identity Verification

Credential Management

System Components:

- Credentials class handling
- Public key infrastructure
- Signature verification
- Identity tracking
- Access control

Node Authentication

Verification Process:

1. Checks node credentials
2. Validates signatures
3. Confirms identity blocks
4. Verifies permissions
5. Enables system access

Security Layers

Protection Mechanisms:

- Multi-level verification
- Cryptographic proofs
- Identity confirmation
- Access control
- Audit logging

Operational Security

System Features:

- Continuous monitoring
- Real-time verification
- Automated checks
- Alert systems
- Security logging

Key Advantages:

- Complete protection
- Strong verification
- Clear audit trails
- Future readiness
- Simple management

Security Benefits:

- Robust encryption
- Reliable verification
- Quantum readiness
- Easy administration
- Full auditability

8. Network Reliability and Worker Management

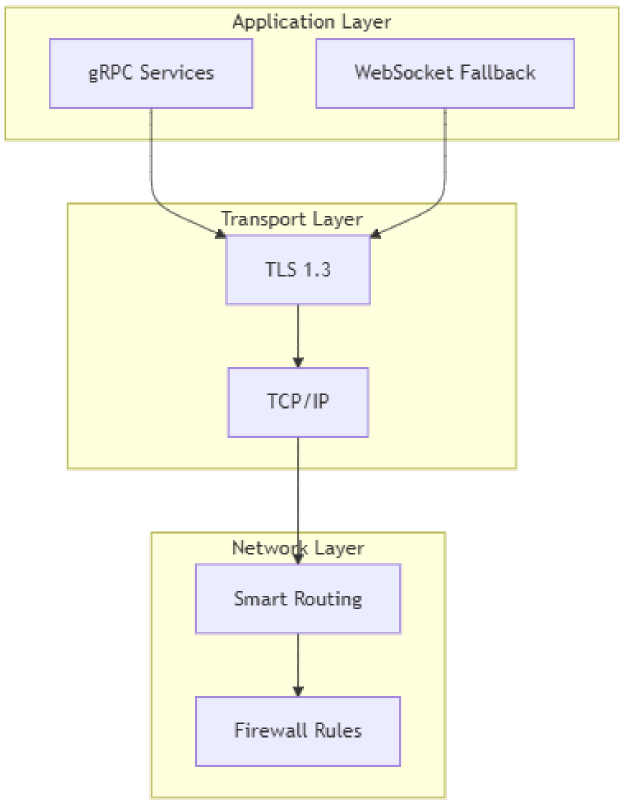

Figure 4: Network protocol implementation showing communication layers, security protocols, and routing systems.

8.1 Worker Expiry and Ping Mechanism

Worker Status Management

Clock Functions:

- Maintains worker registry
- Tracks service addresses
- Monitors node status
- Manages timeouts
- Controls worker states

Timeout Handling

Process Flow:

1. Clock detects inactivity

2. Initiates ping sequence
3. Awaits worker response
4. Updates worker status
5. Logs state changes

Status States

Worker Conditions:

- Active: Fully operational
- Pending: Awaiting response
- Inactive: No response
- Expired: Timeout reached
- Blocked: Security issue

Recovery Process

Reactivation Steps:

1. Worker pushes new block
2. Clock verifies identity
3. Status returns to active
4. Network updates list
5. Operations resume

8.2 Footprint Consistency

Chain Maintenance

Worker Responsibilities:

- Stores local chain copy
- Maintains block sequence
- Verifies transactions
- Updates chain state
- Ensures data integrity

Verification Process

Chain Checks:

- Block sequence validity
- Transaction verification
- Signature confirmation
- Hash consistency
- Timestamp order

Data Access

Availability Features:

- Complete chain history
- Transaction records
- Block verification
- Audit capability
- Quick retrieval

Consistency Management

System Actions:

- Regular state checks
- Automatic updates
- Error detection
- Data validation
- Sync maintenance

Operational Benefits:

- High availability
- Strong consistency
- Quick recovery
- Clear status tracking

- Easy management

Network Advantages:

- Reliable operations
- Fast status updates
- Simple monitoring
- Clear worker states
- Strong data integrity

9. AI and Automation in Sneaky

9.1 SneakyGPT Integration

Core Functionality

Query Capabilities:

- Natural language processing
- Blockchain data access
- Real-time information
- Complex query handling
- Custom result formatting

Technical Implementation

System Components:

- Fine-tuned GPT model
- Blazor interface
- API connections
- Data processors
- Query handlers

Data Access

Query Features:

- Direct chain access
- Real-time updates
- Custom filtering
- Pattern matching
- Result formatting

Security Monitoring

AI Functions:

- Activity pattern analysis
- Anomaly detection
- Threat identification
- Alert generation
- Risk assessment

Business Intelligence

Analysis Features:

- Transaction patterns
- Usage statistics
- Performance metrics
- Trend analysis
- Custom reporting

9.2 Exception Handling Automation

Error Processing

System Actions:

- Detects transaction errors
- Identifies block issues
- Logs exceptions
- Triggers responses

- Updates status

Automated Responses

System Steps:

1. Identifies issue type
2. Applies response rules
3. Executes actions
4. Records outcomes
5. Updates systems

Error Categories

Handled Types:

- Transaction failures
- Block errors
- Network issues
- Security alerts
- Protocol violations

Escalation Protocol

Process Flow:

1. Detects critical issues
2. Evaluates severity
3. Notifies administrators
4. Tracks resolution
5. Updates documentation

Monitoring Systems

Automation Features:

- Real-time tracking

- Status updates
- Performance monitoring
- System health checks
- Resource management

Operational Benefits:

- Quick issue detection
- Automated responses
- Clear tracking
- Easy management
- Strong documentation

System Advantages:

- Intelligent monitoring
- Fast response times
- Reliable handling
- Clear processes
- Full automation

10. Open Source and Community Engagement

10.1 Apache 2.0 Licensing

License Features

Key Permissions:

- Free code use
- Modification rights
- Distribution freedom
- Patent rights
- Commercial use

License Requirements

User Obligations:

- Maintain copyright notices
- Document changes
- Include license copy
- State modifications
- Preserve attributions

Legal Protection

Built-in Safeguards:

- Patent protection
- Trademark rights
- Liability limitations
- Warranty disclaimers
- Usage clarity

10.2 Community Contributions

Development Guidelines

Code Standards:

- Style requirements
- Documentation rules
- Testing protocols
- Review processes
- Quality checks

Contribution Process

Submission Steps:

1. Fork repository

2. Create branch
3. Make changes
4. Submit pull request
5. Address reviews

Code Review

Quality Checks:

- Style compliance
- Test coverage
- Performance impact
- Security review
- Documentation quality

Technical Standards

Development Requirements:

- Clean code practices
- Comprehensive testing
- Clear documentation
- Efficient algorithms
- Security awareness

Community Support

Available Resources:

- Technical guides
- Development tools
- Code examples
- Best practices
- Support channels

Collaboration Tools

Platform Features:

- Issue tracking
- Version control
- Documentation wiki
- Discussion forums
- Project boards

Development Benefits:

- Clear guidelines
- Ready support
- Easy collaboration
- Quality control
- Fast feedback

Community Advantages:

- Open development
- Shared knowledge
- Continuous improvement
- Clear processes
- Strong governance

11. Obvious Best Practice

Sneaky implements recognized industry standards. At the same time, we seek to identify best practise. Each component follows clear, efficient, and secure design principles.

11.1 Technology Selection

Core Technologies

C# and .NET:

- Enterprise-grade stability
- High performance base
- Strong type safety
- Extensive libraries
- Modular architecture

Front-End Development

Blazor Features:

- Fast load times
- Responsive interfaces
- Server-side rendering
- Component reuse
- Browser compatibility

Data Storage

SQLite Benefits:

- Reliable performance
- Simple maintenance
- Zero configuration
- File-based storage
- ACID compliance

Communication

gRPC Advantages:

- Low latency
- Binary protocols
- Strong typing
- Auto-generated code
- Bidirectional streaming

11.2 Security Implementation

Sneaky implements security through multiple complementary layers. Each layer serves a specific purpose. This approach maintains system simplicity and performance.

Core Encryption Methods

Sneaky builds its security foundation on established cryptographic standards. Elliptic Curve Cryptography (ECC) provides the primary key generation mechanism. This choice balances strong security with efficient processing. Each node generates its keys using ECC. This enables fast and secure digital signatures. The system employs secure hash functions to maintain data integrity. Cryptographically secure random number generation supports key operations.

Data Protection Strategy

The system protects data through end-to-end encryption between nodes. Each node stores its private keys in isolated secure storage. This prevents unauthorized access. Access controls operate at both node and data levels. The system maintains comprehensive audit logs of all security events. This enables quick detection of potential issues. Automated intrusion detection monitors for unusual patterns. It also identifies unauthorized access attempts.

Future-Ready Security

Sneaky includes optional quantum-resistant encryption capabilities. This feature helps enterprises prepare for future security threats. It maintains current performance levels. The system supports flexible algorithm selection. This enables updates as security standards evolve. Migration paths ensure smooth transitions between security protocols. Version control tracks all security configuration changes. This maintains system stability during updates.

All security features maintain Sneaky's core principle of simplicity. Each mechanism serves a clear purpose. The system avoids unnecessary complexity. It ensures robust protection. This approach delivers enterprise-grade security without complex blockchain overhead.

11.3 Content Management

Data Handling

Content Features:

- Format independence
- String-based storage
- Flexible payloads
- Easy validation
- Simple parsing

Business Adaptability

System Benefits:

- Multiple use cases
- Format flexibility
- Easy integration
- Simple adoption
- Clear workflows

11.4 System Architecture

Module Design

Structure Benefits:

- Clear role separation
- Simple maintenance
- Easy scaling
- Quick debugging
- Strong isolation

Component Roles

Clear Boundaries:

- Clock functions
- Worker tasks
- Sender operations
- Data flow
- Security checks

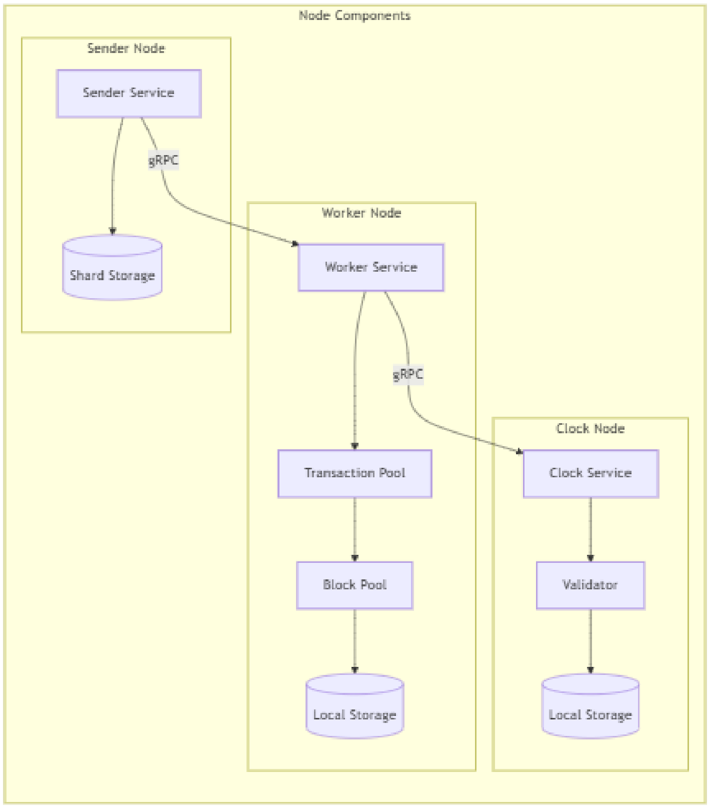

Figure 5: Detailed architecture showing internal components of each node type. Demonstrates storage systems, service layers, and communication protocols.

11.5 Error Management

System Monitoring

Key Features:

- Automated logging
- Real-time alerts
- Issue tracking
- Performance monitoring
- Resource management

Problem Resolution

Process Flow:

1. Error detection
2. Classification
3. Response selection
4. Action execution
5. Resolution tracking

11.6 System Growth

Scalability Features

Design Elements:

- Efficient protocols
- Resource optimization
- Load handling
- Easy expansion
- Performance monitoring

Future Planning

Growth Support:

- Modular design
- Version control
- Update paths
- Feature flags
- Backward compatibility

11.7 Community Focus

Open Development

Key Aspects:

- Clear documentation
- Simple navigation
- Easy contributions

- Version tracking
- Knowledge sharing

Code Quality

Standards:

- Style guidelines
- Testing requirements
- Review processes
- Security checks
- Performance metrics

12. Conclusion

12.1 Summary of Innovations

Technical Achievements

Core Innovations:

- High-performance processing
- Strong security model
- Minimal energy usage
- Clock-based architecture
- Rapid transaction handling

Architecture Benefits

Key Advantages:

- No consensus overhead
- Quick block verification
- Simple node management
- Clear security model
- Easy scaling

System Features

Technical Capabilities:

- Content-agnostic storage
- Post-quantum readiness
- Open-source access
- AI integration
- Natural language queries

12.2 Future Directions

Scaling Improvements

Planned Updates:

- Multi-clock support
- Enhanced sharding
- Increased throughput
- Better load handling
- Performance optimization

Technical Roadmap

Development Focus:

- Security enhancements
- Protocol updates
- Tool improvements
- API expansion
- Feature additions

Growth Areas

Future Capabilities:

- Advanced AI features

- Voice interactions
- Custom explorers
- Industry tools
- Integration options

System Evolution

Development Goals:

- Maintain simplicity
- Enhance security
- Improve performance
- Add features
- Support growth

Get Started

For technical discussions or development inquiries:

- Email: sneakylabs@proton.me
- Documentation: [Available on request]
- Code access: [Repository details]

Bibliography

Bernstein, D. J., Buchmann, J., & Dahmen, E. (2009). *Post-Quantum Cryptography*. Springer.

Berkhout, F., Smith, A., & Stirling, A. (2004). Socio-Technological Regimes and Transition Contexts. In B. Elzen, F. W. Geels, & K. Green (Eds.), *System Innovation and the Transition to Sustainability*. Edward Elgar Publishing.

Boneh, D., & Shoup, V. (2020). *A Graduate Course in Applied Cryptography*. Retrieved from https://toc.cryptobook.us.

Bostrom, N. (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.

Buterin, V. (2015). *On Sharding Blockchains*. Retrieved from https://github.com/ethereum/wiki/wiki/Sharding-FAQs.

Castro, M., & Liskov, B. (1999). Practical Byzantine Fault Tolerance. *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 173-186.

Deloitte Insights. (2019). *Breaking Blockchain Open: Deloitte's 2019 Global Blockchain Survey*. Retrieved from https://www2.deloitte.com/global/en/insights/topics/understanding-blockchain-potential.html.

Drescher, D. (2017). *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Apress.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

IBM Institute for Business Value. (2018). *Building your Blockchain Advantage: How to Start Planning for Blockchain's Business Impact Today*. IBM.

Jiang, P., Wu, J., Chen, J., & Zhao, S. (2019). Blockchain-Based Distributed Energy Trading for Sustainable Development: A Game-Theoretic Approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Kwon, J. (2014). *Tendermint: Consensus without Mining*. Retrieved from https://tendermint.com/static/docs/tendermint.pdf.

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 382-401.

Li, X., Jiang, P., Chen, T., Luo, X., & Wen, Q. (2020). A Survey on the Security of Blockchain Systems. *Future Generation Computer Systems*, 107, 841-853.

Mougayar, W. (2016). *The Business Blockchain: Promise, Practice, and the Application of the Next Internet Technology*. Wiley.

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Retrieved from https://bitcoin.org/bitcoin.pdf.

National Institute of Standards and Technology. (2020). *Post-Quantum Cryptography: NIST's Plan for the Future*. Retrieved from https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

Peters, G. W., Panayi, E., & Chapelle, A. (2015). Trends in Blockchain Technology and Security. In *Handbook of Blockchain, Digital Finance, and Inclusion* (Vol. 1, pp. 241-265). Academic Press.

Pongnumkul, S., Siripanpornchana, C., & Thajchayapong, S. (2017). Performance Analysis of Private Blockchain Platforms in Varying Workloads. *26th International Conference on Computer Communications and Networks (ICCCN)*, 1-6.

Raymond, E. S. (1999). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media.

Risius, M., & Spohrer, K. (2017). A Blockchain Research Framework: What We (Don't) Know, Where We Go from Here, and How We Will Get There. *Business & Information Systems Engineering*, 59(6), 385-409.

Tapscott, D., & Tapscott, A. (2016). *Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*. Portfolio.

Tanenbaum, A. S., & van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. Prentice Hall.

Von Krogh, G., Spaeth, S., & Lakhani, K. R. (2003). Community, Joining, and Specialization in Open Source Software Innovation: A Case Study. *Research Policy*, 32(7), 1217-1241.