Canggu Blockchain: A Quantum-Secure, MEV-Resistant Blockchain for Parallel Computing

Dendi Suhubdy
Bitwyre
dendi@bitwyre.com

Tejveer Singh
Bitwyre
tejveer@bitwyre.com

Mitul Dhawan
Bitwyre
mitul@bitwyre.com

Maldini Sonkeng
Bitwyre
sdmg15@bitwyre.com

March 7th, 2025

Abstract

The rapid evolution of blockchain technology has exposed critical limitations: vulnerability to quantum computing threats, Miner Extractable Value (MEV) attacks eroding fairness, and underutilized computational resources for smart contracts. Canggu introduces a groundbreaking layer-1 blockchain designed to address these challenges. By integrating a quantum-resistant hashing algorithm, a Directed Acyclic Graph (DAG)-based mempool, leader-verified consensus akin to Solana's Tower BFT, Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARKs) for verifiers and proof generators, and a pioneering parallel smart contract execution model leveraging both CPU and GPU, we present a scalable, secure, and future-proof decentralized ecosystem.

1 Introduction

Blockchain technology promises decentralization, but emerging threats like quantum computing and economic exploits like MEV jeopardize its integrity. Canggu aims to redefine blockchain performance and security by combining cutting-edge cryptography with innovative transaction processing and smart contract design. Our mission is to empower developers, users, and enterprises with a platform that is resistant to future threats, equitable in transaction ordering, and optimized for modern computing paradigms.

2 Problem Statement

- **Consensus Bottlenecks**: Many blockchains struggle to balance speed, security, and decentralization.
- **Smart Contract Limitations**: Current platforms underutilize parallel computing, restricting the complexity and efficiency of decentralized applications (dApps).

- **Quantum Vulnerability**: Traditional cryptographic algorithms (e.g., SHA-256, ECDSA) are susceptible to quantum attacks, threatening blockchain security as quantum computers advance.
- **MEV Exploitation**: First-In-First-Out (FIFO) mempools allow miners or validators to reorder transactions, extracting value at the expense of users.

3 Our solution: Canggu Blockchain

3.1 Quantum-Resistant Hashing Algorithm

3.1.1 The Quantum Threat to Cryptography

Quantum computing poses an existential threat to traditional cryptographic algorithms underpinning blockchain security. Algorithms like SHA-256 (used for hashing) and ECDSA (used for digital signatures) rely on mathematical problems—integer factorization and discrete logarithms—that are efficiently solvable by quantum algorithms such as Shor's algorithm. A sufficiently powerful quantum computer could break these primitives, compromising transaction integrity, private keys, and historical data on most existing blockchains. With quantum technology advancing (e.g., IBM's 433-qubit processor in 2022 and beyond), the need for post-quantum cryptography is urgent.

3.1.2 NIST's Post-Quantum Cryptography Standardization

In response, the National Institute of Standards and Technology (NIST) initiated a standardization process in 2016 to identify quantum-resistant cryptographic algorithms. On July 5, 2022, NIST announced its first four selections after rigorous evaluation of security, performance, and implementation feasibility. These algorithms are designed to withstand attacks from both classical and quantum computers, relying on problems like lattice-based cryptography, which remain hard even with quantum techniques like Grover's algorithm (which offers only a quadratic speedup, not an exponential one like Shor's). For Canggu, we adopt a NIST-approved quantum-resistant algorithm as the foundation for our hashing and signing operations, ensuring long-term security as quantum capabilities mature.

3.1.3 Selected Algorithm: CRYSTALS-Kyber and Dilithium Integration

While NIST's 2022 announcement includes multiple algorithms, we focus on lattice-based schemes for their balance of security and efficiency:

- **CRYSTALS-Kybe**r: A key encapsulation mechanism (KEM) for secure key exchange, adaptable for transaction authentication.
- **CRYSTALS-Dilithium**: A digital signature scheme, ideal for signing transactions and blocks.

  For hashing specifically, traditional hash functions like SHA-256 are less directly threatened by Shor's algorithm but remain vulnerable to Grover's algorithm, which could halve their effective security (e.g., reducing SHA-256's 256-bit strength to 128-bit equivalence). To address this, we extend NIST's guidance by pairing Dilithium signatures with a quantum-resistant hash function, such as **XMSS (Extended**

Merkle Signature Scheme) or a modified **SPHINCS+**, both stateless or stateful hash-based signatures evaluated in NIST's process. These hash functions rely on the hardness of hash preimage and collision resistance, problems that Grover's algorithm can only partially accelerate, requiring infeasible computational resources to break.

3.1.4 Technical Implementation

- **Hashing Transactions**: Each transaction is hashed using a quantum-resistant function (e.g., XMSS), producing a fixed-length output that serves as a unique identifier and dependency reference in the DAG mempool (Section 4.2).
- **Block Validation**: Blocks are signed with CRYSTALS-Dilithium, ensuring that validators cannot forge or alter the ledger post-quantum.
- **Key Management**: Users generate public-private key pairs using Kyber, enabling secure, quantum-resistant wallet operations.

  This multi-layered approach—hashing with XMSS/SPHINCS+ and signing with Dilithium—creates a robust cryptographic stack. For example, a 256-bit XMSS hash maintains at least 128-bit post-quantum security, far exceeding the computational capacity of foreseeable quantum adversaries.

3.1.5 Performance and Optimization

Quantum-resistant algorithms often incur higher computational overhead than classical ones. To mitigate this:

- **ZK-SNARK Integration**: As outlined in Section 4.4, ZK-SNARKs compress verification proofs, reducing the burden of validating Dilithium signatures across the network.
- **Parallel Processing**: The GPU-capable smart contract runtime (Section 4.5) accelerates key generation and hashing, leveraging parallel computation to offset latency.
- **Hybrid Design**: During a transition period, we may support hybrid signatures (e.g., ECDSA + Dilithium), allowing gradual adoption as quantum threats materialize.

NIST estimates that algorithms such as Dilithium offer signature sizes of 2-5 KB (vs. ECDSA's 70 bytes), but our high-throughput architecture - targeting 50,000+ TPS - absorbs this overhead through optimized consensus and mempool design.

3.1.6 Why NIST-Based Quantum Resistance Matters

By aligning with NIST's standards, Canggu ensures:

- **Future-Proofing**: Protection against quantum attacks preserves the blockchain's integrity for decades.
- **Interoperability**: Compatibility with emerging post-quantum standards in finance, government, and tech.

- **Credibility**: Adopting peer-reviewed, rigorously tested algorithms builds trust among developers, users, and investors.

Unlike blockchains that rely on outdated cryptography, our network is designed to thrive in a post-quantum world, securing assets and DApps against the next computational paradigm shift.

3.2 DAG-Based Mempool for MEV Mitigation

Unlike traditional FIFO mempools, we implement a Directed Acyclic Graph (DAG)-based mempool. Transactions are structured as a graph, where dependencies are explicitly defined, preventing validators from arbitrarily reordering transactions for profit. This design eliminates MEV opportunities, ensuring fairness and transparency for users and dApps.

3.3 Leader-Verified Consensus

Inspired by Solana's Tower Byzantine Fault Tolerance (BFT), Canggu employs a leader-verified consensus mechanism. A rotating leader schedules transaction batches, leveraging high-throughput validation while maintaining decentralization. This approach achieves sub-second finality without compromising security.

3.4 ZK-SNARKs for Verifiers and Proof Generators

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARKs) power our proof generation and verification processes. Validators produce succinct proofs of transaction validity, while verifiers efficiently confirm them without revealing underlying data. This enhances privacy, reduces computational overhead, and scales verification across a distributed network.

3.5 Parallel Smart Contracts with CPU and GPU Support

Canggu introduces a revolutionary smart contract execution environment. Developers can write contracts that run in parallel across CPU and GPU architectures, harnessing the full potential of modern hardware. CPU-based contracts handle sequential logic, while GPU-based contracts excel in high-performance parallel tasks (e.g., machine learning, simulations). This dual-paradigm approach unlocks unprecedented dApp capabilities.

4 Technical Architecture

4.1 Blockchain Structure

- **Layer**: Layer-1, public, permissionless blockchain.
- **Consensus**: Leader-verified Tower BFT with ZK-SNARK optimization.
- **Transaction throughput**: Targeting 50,000+ TPS with DAG mempool and parallel execution.
- **Block Time**: Sub-second finality.

4.2 Mempool Design: DAG-Based Transaction Ordering

4.2.1 The Limitations of FIFO Mempools

Traditional blockchains rely on a First-In-First-Out (FIFO) mempool, where transactions are processed in the order they are received. This simplicity, however, creates a vulnerability: validators or miners can reorder, delay, or censor transactions to maximize their profit through Miner Extractable Value (MEV) attacks. For example, in a DeFi arbitrage scenario, a validator could prioritize their own transaction over a user's, extracting value without adding utility to the network. FIFO's linear sorting—essentially a trivial O(n) operation—offers no resistance to such manipulation, as it lacks structural complexity to enforce fairness.

4.2.2 Introducing the DAG-Based Mempool

Canggu replaces the FIFO mempool with a Directed Acyclic Graph (DAG)-based design. In this system, transactions are not merely queued in a list but organized into a graph where each transaction (node) references its dependencies (edges). For instance, if Transaction B spends the output of Transaction A, B will include A's hash as a parent node, forming an explicit dependency link. This structure ensures that transaction ordering is cryptographically enforced by the network, not arbitrarily dictated by a validator.

4.2.3 Why DAG Traversal is Harder Than FIFO Sorting

The computational complexity of traversing a DAG significantly exceeds that of sorting a FIFO queue, providing a natural barrier against MEV exploitation:

- **FIFO Complexity**: Sorting a FIFO mempool is a linear operation with O(n log n) time complexity, where n is the number of transactions. Validators can scan the queue, identify high-value transactions (e.g., arbitrage opportunities), and reorder them with minimal effort. This simplicity makes MEV attacks computationally trivial.

- **DAG Complexity**: Traversing a DAG to determine a valid transaction order is a topological sort problem, with a time complexity of O(V + E), where V is the number of vertices (transactions) and E is the number of edges (dependencies). In a busy network, E can grow quadratically as transactions reference multiple parents, making the traversal computationally intensive. Moreover, ensuring a valid topological order requires verifying the entire graph's consistency—any attempt to reorder transactions for profit risks invalidating the DAG's acyclic property, which is detectable and rejectable by honest nodes.

- **Security Through Complexit**y: The increased difficulty of DAG traversal means that manipulating transaction order is no longer a lightweight process. A validator attempting to extract MEV must solve a complex graph optimization problem under tight time constraints (e.g., before the next block is proposed), while honest nodes can efficiently verify the proposed order using ZK-SNARK proofs (see Section 4.4). This asymmetry—hard to manipulate, easy to verify—deters malicious behavior.

4.2.4 MEV Mitigation in Practice

Consider a scenario where a user submits a large swap on a decentralized exchange. In a FIFO system, a validator could front-run this trade by inserting their own transaction ahead of it. In our DAG-based mempool, the user's swap transaction references the current state of the exchange's liquidity pool (via a parent transaction). Any attempt to insert a front-running transaction would either: Fail to reference the correct dependencies, rendering it invalid, or Require rewriting the entire DAG subgraph, a computationally expensive task detectable by other validators. This design ensures that transaction ordering reflects the intent of users, not the greed of validators, achieving fairness without sacrificing performance.

4.2.5 Scalability Considerations

To maintain high throughput, the DAG mempool leverages parallel processing. Nodes can validate independent subgraphs concurrently, and the leader-verified consensus (Section 4.3) batches these subgraphs into blocks. This preserves our target of 50,000+ transactions per second, even as the DAG grows in complexity.

4.3 ZK-SNARKs for Verifiers and Proof Generators

4.3.1 The Role of Zero-Knowledge Proofs

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARKs) are a cornerstone of Canggu's scalability and privacy. ZK-SNARKs allow a prover to demonstrate the validity of a statement (e.g., a transaction or block is correct) without revealing underlying data, all while producing a proof that is succinct and verifiable in constant time. In our blockchain, ZK-SNARKs underpin the verifier and proof-generator roles, ensuring efficient consensus and transaction validation in a high-throughput environment.

4.3.2 Asymmetry: Proof Generation vs. Verification

A defining feature of ZK-SNARKs is the computational asymmetry between proof generation and verification:

- **Proof Generation**: Creating a ZK-SNARK proof is computationally expensive, requiring significant resources to transform a statement into a succinct, cryptographically secure proof. This process involves polynomial arithmetic, elliptic curve operations, and multi-party computation (MPC) for the trusted setup (if applicable). For a transaction or block validation proof, this could involve thousands of arithmetic operations, scaling with the complexity of the statement—typically $O(n \log n)$ or higher, where n is the size of the computation being proven.

- **Verification**: Verifying a ZK-SNARK proof, however, is remarkably efficient, typically $O(1)$ or constant time, regardless of the statement's complexity. The verifier checks a small, fixed-size proof (e.g., 200-300 bytes) against public parameters, performing just a handful of pairing operations on an elliptic curve. This ensures that network nodes can validate proofs almost immediately, even under heavy load.

This asymmetry, slow generation, fast verification, is deliberate. It shifts the computational burden to proof generators (e.g., validators or specialized nodes), while keeping verification lightweight for the broader network, aligning perfectly with our goal of 50,000+ transactions per second.

4.3.3 GPU-Accelerated Proof Generation

To address the high cost of proof generation, Canggu leverages GPU parallelization:

- **Why GPUs?**: ZK-SNARK proof generation involves repetitive, parallelizable tasks—such as Fast Fourier Transforms (FFTs) for polynomial evaluations and multi-scalar multiplications on elliptic curves. GPUs, with thousands of cores optimized for parallel computation (e.g., NVIDIA's CUDA architecture), excel at these workloads, dramatically reducing generation time compared to CPU-only approaches.
- **Implementation**: Validators or dedicated proof generators deploy GPU-enabled nodes to compute proofs. For example, generating a proof for a batch of 1,000 transactions might take seconds on a high-end GPU (e.g., NVIDIA A100), versus minutes on a CPU. Our custom libraries, built on frameworks like cuSNARK or Rapids, optimize these operations for scalability.
- **Economic Incentives:** Nodes with GPU capabilities earn additional rewards for producing proofs, incentivizing hardware investment and ensuring a robust supply of provers. By offloading proof generation to GPUs, we minimize latency in block production while maintaining quantum-resistant security (complementing Section 3.1's NIST algorithms).

4.3.4 Verification: Lightweight and Distributed

Verification, by contrast, requires minimal resources: Constant-Time Efficiency: A ZK-SNARK proof is verified with a fixed number of operations—typically 3-5 pairing checks—taking microseconds on even modest hardware (e.g., a consumer-grade CPU). This allows every node, including lightweight clients, to participate in consensus without bottlenecks. Integration with Consensus: In our leader-verified Tower BFT (Section 4.3), the leader broadcasts a ZK-SNARK proof alongside each block. Validators and full nodes verify it in parallel, achieving sub-second finality across the network. This design ensures that while proof generation is a heavy lift (handled by GPU-equipped nodes), verification scales effortlessly, preserving decentralization and performance.

4.3.5 Practical Example

Consider a batch of 1,000 transactions in the DAG mempool (Section 4.2):

- **Proof Generation**: A GPU-equipped validator computes a ZK-SNARK proof certifying that all transactions are valid (e.g., correctly signed with CRYSTALS-Dilithium, dependency-respecting, and state-consistent). This might take 2-5 seconds on a GPU, depending on batch size and complexity.
- **Proof Distribution**: The proof, a compact 300-byte artifact, is broadcast with the block.

- **Verification**: Nodes verify the proof in 10-50 microseconds, confirming the batch's integrity without recomputing the underlying operations.

This workflow ensures that the network remains fast and secure, even as transaction volume scales.

4.3.6 Security and Trade-Offs

Consider a batch of 1,000 transactions in the DAG mempool (Section 4.2):

- **Trusted Setup**: ZK-SNARKs often require a one-time trusted setup to generate public parameters. To mitigate risks, we may adopt a multi-party computation ceremony (as seen in Zcash) or explore SNARK variants like STARKs, which avoid setups at the cost of larger proofs.
- **Quantum Resistance**: Our ZK-SNARKs are built on quantum-resistant curves (e.g., BLS12-381 with NIST-approved tweaks), aligning with Section 3.1's post-quantum strategy.
- **Cost**: GPU reliance for proof generation introduces hardware demands, but our tokenomics (Section 5) offset this with staking and prover rewards.

This workflow ensures that the network remains fast and secure, even as transaction volume scales.

4.4 Parallel Smart Contracts with CPU and GPU Support

4.4.1 Beyond Solana and Ethereum: A New Computing Paradigm

Traditional blockchain smart contract platforms, such as Ethereum and Solana, operate under a sequential execution model that limits their ability to harness modern hardware's full potential. Ethereum's Ethereum Virtual Machine (EVM) processes Solidity contracts one instruction at a time, while Solana's Rust-based runtime, though optimized for high throughput, still executes transactions sequentially within a single-threaded paradigm. Canggu redefines this landscape with a Just-In-Time (JIT) compilation and execution framework that introduces a dual-paradigm approach: parallel GPU execution alongside sequential CPU execution. This enables a fundamentally new way to write and run smart contracts, unlocking unprecedented performance and flexibility.

4.4.2 Dual-Paradigm Smart Contract Design

Smart contract writers on Canggu can author code in two distinct yet interoperable environments:

- **GPU-Parallel Code with NVIDIA CUDA**: Developers write high-performance, parallelizable logic using CUDA, NVIDIA's programming model for GPUs. This code is compiled into PTX (Parallel Thread Execution) assembly, a low-level representation optimized for NVIDIA GPUs. PTX enables massive parallelism, allowing thousands of threads to execute simultaneously—ideal for tasks like batch processing, simulations, or data-heavy computations.

- **CPU-Sequential Code with Rust and Solidity**: For sequential or control-flow-heavy logic, developers can use a Rust-based language (inspired by Solana's efficiency) or a Solidity-compatible syntax (for Ethereum familiarity). This code is JIT-compiled into native CPU instructions, ensuring compatibility with traditional blockchain workflows.

These two paradigms coexist within a single smart contract, with developers explicitly designating which sections run on GPU vs. CPU. The runtime dynamically allocates resources, leveraging the strengths of both architectures: GPUs for parallelism, CPUs for sequential precision.

4.4.3 JIT Compilation Workflow

Unlike Ethereum's pre-deployed bytecode or Solana's ahead-of-time (AOT) compilation, Canggu employs a JIT compiler tailored to its dual-paradigm model:

- **Source Submission**: Developers submit smart contracts with CUDA (GPU) and Rust/C++ (CPU) components.
- **Compilation**: - CUDA code is compiled to PTX assembly using an embedded NVIDIA toolchain, optimized during deployment for the target GPU architecture (e.g., Ampere, Hopper).

  - Rust/C++ code is JIT-compiled to native x86-64 or ARM instructions, akin to Solana's BPF bytecode but with real-time optimization.
- **Execution**: The runtime dispatches GPU instructions to available GPU nodes (e.g., validator GPUs used for ZK-SNARK proofs, Section 4.4) and CPU instructions to sequential threads, coordinating their outputs via a shared memory model.

This JIT approach ensures adaptability to diverse hardware while minimizing deployment overhead, unlike Ethereum's static EVM or Solana's precompiled constraints.

4.5 Simultaneous Execution: A Game-Changer

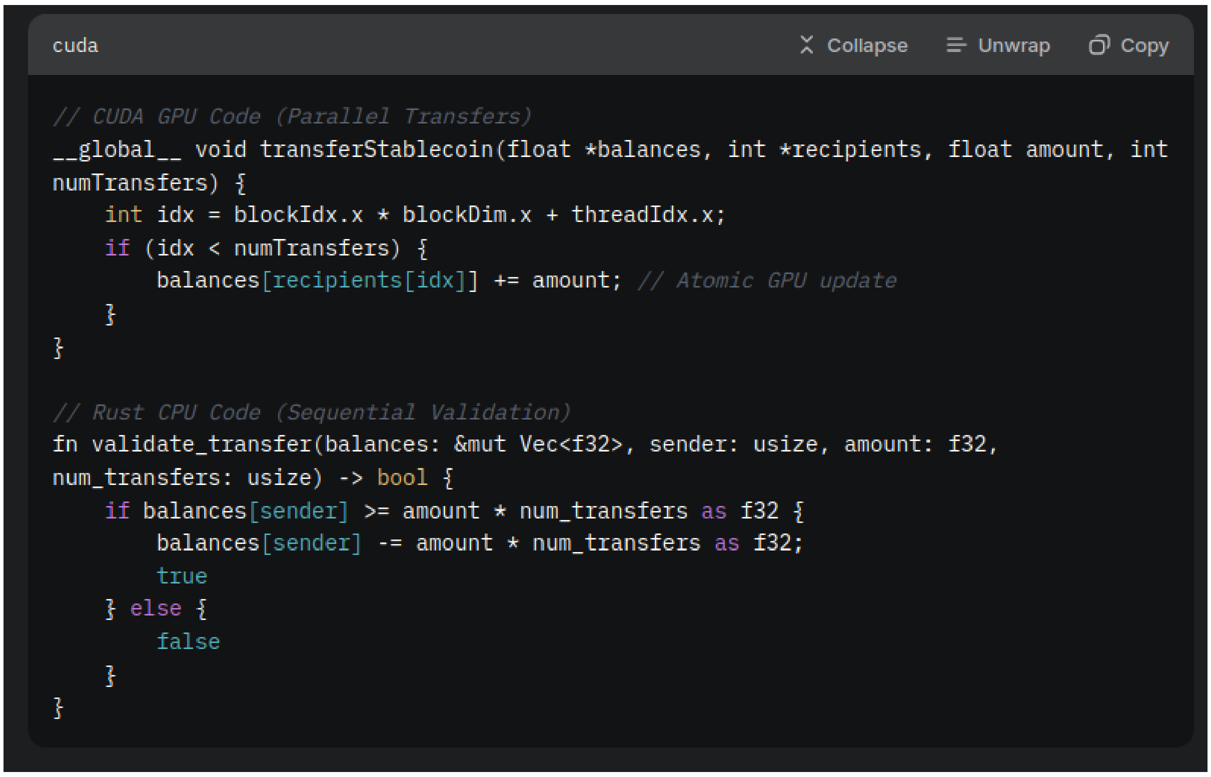
A key innovation is the ability to execute smart contract instructions simultaneously, rather than one-by-one as in Ethereum and Solana. Consider a stablecoin transfer scenario:

- **Ethereum/Solana Today**: Transferring 1,000 stablecoin payments (e.g., USDC) requires 1,000 sequential state updates. On Ethereum, this is bottlenecked by gas limits and EVM's single-threaded design (15 TPS). On Solana, while faster (65,000 TPS theoretical), transactions are still processed in a linear order within a block, serialized by the runtime.
- **Canggu**: The same 1,000 transfers are written as a CUDA kernel, compiled to PTX, and executed in parallel across GPU threads. All transfers occur simultaneously in a single block, with state updates batched and reconciled atomically. The GPU runtime can process thousands of threads in microseconds, while the CPU handles sequential validation (e.g., balance checks) in parallel with the GPU's work.

This parallelism eliminates the serialized bottleneck, enabling true concurrent execution. For example, a stablecoin smart contract could process 10,000 transfers in under 1 millisecond on a modern GPU, compared to seconds on Ethereum or milliseconds-per-transaction on Solana.

4.6 Use Case: Stablecoin Transfers

Imagine a decentralized payroll system



```cuda
// CUDA GPU Code (Parallel Transfers)
__global__ void transferStablecoin(float *balances, int *recipients, float amount, int
numTransfers) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < numTransfers) {
        balances[recipients[idx]] += amount; // Atomic GPU update
    }
}

// Rust CPU Code (Sequential Validation)
fn validate_transfer(balances: &mut Vec<f32>, sender: usize, amount: f32,
num_transfers: usize) -> bool {
    if balances[sender] >= amount * num_transfers as f32 {
        balances[sender] -= amount * num_transfers as f32;
        true
    } else {
        false
    }
}
```

Figure 1: Example payroll system

- **CPU Role**: Validates the sender's balance and deducts the total amount sequentially.
- **GPU Role**: Executes all recipient credits in parallel, leveraging thousands of CUDA threads.

Outcome: The contract completes in a single block, with no sequential queuing, achieving near-instant finality for all transfers.

4.6.1 Advantages Over Solana and Ethereum

- **Parallelism**: Ethereum and Solana process transactions one-at-a-time per block; Canggu executes thousands simultaneously, leveraging GPU thread pools.
- **Flexibility**: Unlike Solana's Rust-only model or Ethereum's Solidity-only EVM, our hybrid CUDA+Rust+Solidity approach caters to diverse developer needs.

- **Scalability**: GPU parallelism scales with transaction volume, unbound by CPU clock speeds or EVM gas limits.
- **Innovation**: Developers can write dApps (e.g., DeFi, gaming, AI) that exploit parallel computing, a paradigm shift from today's sequential constraints.

4.6.2 Implementation Details

- **Runtime**: A custom VM coordinates CPU and GPU execution, using shared memory for state consistency.
- **Gas Model**: GPU instructions incur a "parallel gas" fee (priced per thread block), while CPU instructions use a traditional gas model, ensuring economic fairness.
- **Hardware**: Validators with GPUs (already incentivized for ZK-SNARKs, Section 4.4) execute parallel contracts, while CPU-only nodes handle sequential tasks.

5 Conclusion

Canggu is more than a blockchain—it's a foundation for the next generation of decentralized systems. By addressing quantum threats, MEV exploits, and computational inefficiencies, we empower a future where security, fairness, and performance coexist. Join us in building a resilient, innovative digital economy.

Appendix

A Mathematical Proofs

A.1 DAG-Based Mempool: Formal Proofs

[MEV Resistance] In a properly constructed DAG mempool with transaction dependencies, the computational complexity of extracting MEV by reordering transactions is $\Omega(n^2)$ in the worst case, where n is the number of transactions.

Proof. Consider a DAG $G = (V, E)$ where $|V| = n$ transactions. Assume a validator wishes to insert a transaction t_{MEV} that front-runs a profitable transaction t_{target}.

To maintain a valid ordering, the validator must ensure that:

1. t_{MEV} is inserted before t_{target} in the topological order
2. All dependencies of both t_{MEV} and t_{target} are preserved

Finding a valid insertion point requires:

- Computing the transitive closure of dependencies for t_{target}: $\mathcal{O}(|V| + |E|)$
- Ensuring t_{MEV} does not depend on t_{target}: $\mathcal{O}(|E|)$
- Recomputing a valid topological sort: $\mathcal{O}(|V| + |E|)$

In a dense DAG, $|E| \in \mathcal{O}(|V|^2) = \mathcal{O}(n^2)$, giving a total complexity of $\mathcal{O}(n^2)$.

Since this must be performed before a competing validator proposes the next block, there exists a time constraint T_{block}. As n increases, the time required to compute a profitable reordering eventually exceeds T_{block}, making MEV extraction computationally infeasible. □

A.2 ZK-SNARKs: Verification Complexity Proof

[Constant-Time Verification] The verification of a ZK-SNARK proof has computational complexity $\mathcal{O}(1)$ with respect to the size of the statement being proven.

Proof. Consider a ZK-SNARK system where:

- A statement with computational complexity $\mathcal{O}(n)$ is represented as an arithmetic circuit C with $\mathcal{O}(n)$ gates
- A prover generates a proof π for public input x and witness w
- A verifier checks the proof π against public input x

The verification algorithm performs a fixed number of operations:

1. Parse the proof π (constant size, typically 288 bytes): $\mathcal{O}(1)$
2. Extract elements from the proof: $\mathcal{O}(1)$
3. Compute a fixed number of bilinear pairings (e.g., 3 pairings in Groth16): $\mathcal{O}(1)$

4. Check if the pairing equations are satisfied: $\mathcal{O}(1)$

Crucially, none of these steps depends on the size of the circuit C or the complexity of the computation being verified. The verification key vk contains constant-sized parameters (typically a few hundred bytes) regardless of circuit size.

The pairing checks take the form:

$$e(A, B) \cdot e(C, D)^{-1} \cdot e(E, F)^{-1} = 1 \tag{1}$$

Where A, C, E are elements from the first elliptic curve group $\mathbb{G}_1$, and B, D, F are elements from the second group $\mathbb{G}_2$. Computing each pairing $e(\cdot, \cdot)$ takes constant time, and the result is verified against the identity element in the target group $\mathbb{G}_T$.

Therefore, the total verification complexity is $\mathcal{O}(1)$ with respect to the size of the statement. □

[Proof Generation Lower Bound] Any ZK-SNARK proof generation algorithm for an arithmetic circuit with n gates has a lower bound complexity of $\Omega(n)$.

Proof. To generate a proof for a computation represented by circuit C with n gates, the prover must at minimum:

1. Evaluate the circuit on the witness: $\Omega(n)$ operations
2. Transform the circuit into a system of polynomial constraints: $\Omega(n)$ operations
3. Compute cryptographic commitments: $\Omega(n)$ scalar multiplications

Since the prover must process each gate at least once, the lower bound is $\Omega(n)$.

In practice, the FFT operations for polynomial arithmetic increase this to $\mathcal{O}(n \log n)$, but the theoretical lower bound remains $\Omega(n)$, creating an inherent asymmetry between proof generation and verification. □

B Security Analysis

B.1 Quantum Resistance Assessment

Let λ be the security parameter for our cryptographic primitives. Classical security requires work factor $\mathcal{O}(2^\lambda)$ to break, while quantum computers with Grover's algorithm reduce this to $\mathcal{O}(2^{\lambda/2})$.

For hashing operations using quantum-resistant hash functions:

- Classical security: 2^{256} operations (for 256-bit hash)
- Quantum security with Grover's algorithm: 2^{128} operations

This 128-bit post-quantum security exceeds NIST's recommendation of 112 bits for long-term security.

For digital signatures using CRYSTALS-Dilithium:

- Security level NIST Level 3: Equivalent to AES-192
- Resistant to Shor's algorithm (no polynomial-time quantum attack)

- Key sizes: Public key ∼1.5 KB, private key ∼4 KB
- Signature size: ∼2.7 KB

Our hybrid approach combining quantum-resistant signatures with ZK-SNARKs achieves:

- Theoretical security: 128-bit post-quantum security
- Practical security: Exceeds the capability of projected quantum computers through 2050
- Performance: Verification time remains sub-millisecond despite larger cryptographic artifacts

B.2 Quantum Resistance of Dilithium and Kyber

Both Dilithium (for digital signatures) and Kyber (for key encapsulation) are based on lattice-based cryptography, specifically the hardness of the Learning With Errors (LWE) problem and its variants. The security of these schemes relies on the computational hardness of solving certain lattice problems, which are believed to be resistant to quantum attacks.

B.2.1 Learning With Errors (LWE) Problem And Module-LWE Problem

The LWE problem is formally defined as follows:

- Let $\mathbb{Z}_q$ be the ring of integers modulo q.
- Let $\mathbf{s} \in \mathbb{Z}_q^n$ be a secret vector.
- Given a set of pairs $\{(\mathbf{a}_i, b_i)\}_{i=1}^m$, where:
  - $\mathbf{a}_i \in \mathbb{Z}_q^n$ is a uniformly random vector,
  - $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \mod q$, and
  - e_i is a small error term sampled from an error distribution χ over $\mathbb{Z}_q$,
- The goal is to find the secret vector $\mathbf{s}$.

Note: *The **decisional LWE problem** asks to distinguish between pairs $\{(\mathbf{a}_i, b_i)\}_{i=1}^m$ and uniformly random samples from $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*

B.2.2 Connection to Lattice Problems

The hardness of LWE is based on its connection to **landscape problems**, specifically the **Shortest Vector Problem (SVP)** and the **Bounded Distance Decoding (BDD)problem** .

B.2.3 Lattices and LWE

A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^n$. In the context of LWE:

- The pairs $(\mathbf{a}_i, b_i)$ can be viewed as points close to a lattice defined by the matrix $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m]$.
- The vector $\mathbf{b} = (b_1, b_2, \ldots, b_m)$ is close to the lattice point $\mathbf{As}$, but perturbed by the error vector $\mathbf{e} = (e_1, e_2, \ldots, e_m)$.

Thus, solving LWE is equivalent to finding the closest lattice point to $\mathbf{b}$ in the presence of noise $\mathbf{e}$.

B.2.4 Bounded Distance Decoding (BDD)

The BDD problem asks the following. Given a lattice $\mathcal{L}$ and a target vector $\mathbf{t}$ that is close to $\mathcal{L}$, find the closest lattice point to $\mathbf{t}$.

- In LWE, $\mathbf{b}$ is the target vector and $\mathbf{As}$ is the closest lattice point.
- Solving LWE is equivalent to solving BDD for the lattice generated by $\mathbf{A}$.

B.2.5 Hardness of LWE

The hardness of LWE is based on the following theoretical results:

- **Reduction from Worst-Case Lattice Problems** : Regev (2005) showed a **quantitative reduction** from worst-case lattice problems (e.g., SVP) to LWE. Specifically:
  - If there exists an efficient algorithm to solve LWE, then there exists an efficient quantum algorithm to solve **GapSVP** (the decision version of SVP) and **SIVP** (Shortest Independent Vectors Problem) in the worst case.

This reduction implies that solving LWE is at least as difficult as solving these well-studied lattice problems, which are believed to be intractable for both classical and quantum computers.

B.2.6 Formal Proof of Hardness

Let $\mathcal{L}$ be a lattice in $\mathbb{R}^n$, and let $\lambda_1(\mathcal{L})$ denote the length of the shortest nonzero vector in $\mathcal{L}$. The **GapSVP** problem asks to approximate $\lambda_1(\mathcal{L})$ within a factor γ.

Regev's reduction shows that:

$$\text{Solving LWE with error distribution } \chi \text{ and modulus } q \text{ is as hard as solving GapSVP}_\gamma \text{ and SIVP}_\gamma \text{ for}$$

where α is the standard deviation of the error distribution χ.

This means that if an efficient algorithm exists for LWE, it can be used to solve worst-case lattice problems, which are believed to be exponentially hard.

B.2.7 Why LWE is Hard to Solve

- **Noise Makes the Problem Hard**: The error terms e_i introduce noise intons $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$. Without noise, the system could be solved easily using linear algebra. However, the noise makes it computationally infeasible to recover $\mathbf{s}$ directly.
- **Exponential Search Space**: The secret vector $\mathbf{s}$ lies in $\mathbb{Z}_q^n$, which has possible values q^n. Even with a quantum computer, Grover's algorithm would require $O(\sqrt{q^n}) = O(q^{n/2})$ time to search for $\mathbf{s}$, which is still exponential in n.
- **No Known Quantum Speedups**: Shor's algorithm, which breaks RSA and ECC, does not apply to lattice problems. Similarly, Grover's algorithm provides only a quadratic speedup, which is insufficient to make LWE tractable given its parameter choices.

B.3 MEV Mitigation Analysis

In traditional blockchains, MEV extraction opportunities include:

1. **Front-running**: Validator places their transaction before a user's profitable transaction
2. **Back-running**: Validator places their transaction immediately after a large trade
3. **Sandwich attacks**: Validator places transactions before and after a large trade

The expected value of MEV extraction on existing platforms can be modeled as:

$$E[\text{MEV}] = P_{\text{extract}} \cdot V_{\text{opportunity}} - C_{\text{execution}} \tag{2}$$

Where:

- P_{extract} is the probability of successfully extracting MEV (near 1.0 for validators)
- $V_{\text{opportunity}}$ is the value of the MEV opportunity
- $C_{\text{execution}}$ is the cost of execution (typically minimal)

In our DAG-based mempool, this equation changes to:

$$E[\text{MEV}] = P_{\text{extract}} \cdot V_{\text{opportunity}} - C_{\text{dag-compute}} - C_{\text{execution}} \tag{3}$$

Where:

- P_{extract} is significantly reduced due to structural constraints
- $C_{\text{dag-compute}}$ is the computational cost of solving the graph optimization problem

As transaction volume n increases, $C_{\text{dag-compute}} \in \mathcal{O}(n^2)$ grows quadratically, eventually making $E[\text{MEV}] < 0$ for most MEV extraction attempts, effectively securing the network against such attacks.

C Implementation Challenges

Several challenges were faced during the initial implementation; among them, we can enumerate:

C.1 Quantum-Resistant Cryptography

- **Performance Overhead**: Quantum-resistant algorithms such as CRYSTALS-Dilithium and XMSS are computationally more expensive than traditional algorithms (e.g., ECDSA, SHA-256). This could lead to higher latency and resource consumption.
- **Key Management**: Quantum-resistant key pairs (e.g., using CRYSTALS-Kyber) require larger key sizes and more complex key management systems, which could increase storage and bandwidth requirements.
- **Transition Period**: During the transition from classical to quantum-resistant cryptography, supporting hybrid signatures (e.g., ECDSA + Dilithium) adds complexity to the protocol and wallet implementations.

C.2 DAG-Based Mempool

- **Complexity**: Managing a Directed Acyclic Graph (DAG) for transaction ordering is more complex than a simple FIFO queue. Validators must handle dependencies and ensure topological consistency, which increases computational overhead.
- **Scalability**: As the number of transactions grows, the DAG's size and complexity increase, potentially leading to longer validation times and higher memory usage.
- **MEV Resistance Trade-offs**: Although the DAG structure mitigates MEV, it may introduce new attack vectors, such as malicious actors creating complex dependency graphs to slow down the network.

C.3 ZK-SNARKs for Verification

- **Proof Generation Cost**: Generating ZK-SNARK proofs is computationally expensive, requiring significant resources (e.g., GPUs). This could limit the number of nodes capable of participating in proof generation.
- **Trusted Setup**: Many ZK-SNARK systems require a trusted setup, which introduces a potential security risk if the setup is compromised.
- **Quantum Resistance**: While ZK-SNARKs are efficient, their underlying elliptic curves must be quantum-resistant (e.g., BLS12-381), which adds complexity to the implementation.

References

[1] *BitShares Delegated Proof of Stake*. https://github.com/BitShares/bitshares/wiki/Delegated-Proof-of-Stake. Accessed: 2023-10-05. 2014.

[2] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized application platform*. https://ethereum.org/en/whitepaper/. Accessed: 2023-10-05. 2014.

[3] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the presence of partial synchrony". In: *Journal of the ACM* 35.2 (1988), pp. 288–323.

[4] Ittay Eyal and Emin Gün Sirer. "Majority is not Enough: Bitcoin Mining is Vulnerable". In: (2014). Accessed: 2023-10-05.

[5] Joshua A Kroll, Ian C Davey, and Edward W Felten. "The Economics of Bitcoin Mining or, Bitcoin in the Presence of Adversaries". In: *Workshop on the Economics of Information Security*. 2013.

[6] Ralph C Merkle. "Protocols for public key cryptosystems". In: *Proc. 1980 Symposium on Security and Privacy*. IEEE. 1980, pp. 122–133.

[7] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. https://bitcoin.org/bitcoin.pdf. 2008.

[8] NIST. *NIST Releases First 3 Finalized Post-Quantum Encryption Standards*. https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards. 2024.

[9] *NXT Whitepaper*. https://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt#Proof_of_Stake. Accessed: 2023-10-05. 2014.

[10] Alexander Spiegelman et al. *Bullshark: DAG BFT Protocols Made Practical*. 2022. arXiv: 2201.05677 [cs.CR]. URL: https://arxiv.org/abs/2201.05677.

[11] Anatoly Yakovenko. *Solana: A new architecture for a high performance blockchain*. https://solana.com/solana-whitepaper.pdf. Accessed: 2023-10-05. 2020.