L05: Blockchain Fundamentals

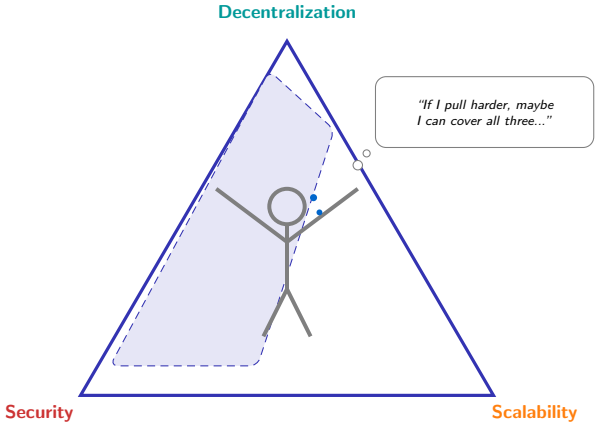
Extended Slides – BSc Digital Finance Course

Digital Finance

What Will You Be Able to Do After This Lecture?

1. Formally define cryptographic hash functions and prove why they make blockchain immutability possible
2. Model Proof of Work and Proof of Stake as games and identify the Nash equilibrium conditions
3. Explain the scalability trilemma as a constrained optimization problem and locate major blockchains on the trade-off surface
4. Implement hash chains, Merkle trees, and PoW simulations in Python
5. Formally model a smart contract as a finite state machine and identify reentrancy as a state transition violation
6. Describe zero-knowledge proof systems and evaluate their performance trade-offs (proof size, verification time, trusted setup)

This lecture goes deep: formal math, working Python code, and 11 data visualizations. By the end you will understand blockchain from the inside out.



The scalability trilemma: every blockchain designer's impossible bedtime.

Why Is a Hash Function the Foundation of All Blockchain Security?

Definition. A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ satisfies:

Pre-image resistance:	Given h, infeasible to find m s.t. $H(m) = h$
Second pre-image:	Given m_1, infeasible to find $m_2 \neq m_1$ s.t. $H(m_1) = H(m_2)$
Collision resistance:	Infeasible to find any (m_1, m_2) with $H(m_1) = H(m_2)$

Avalanche effect: For $m' = m \oplus e_j$ (single-bit flip):

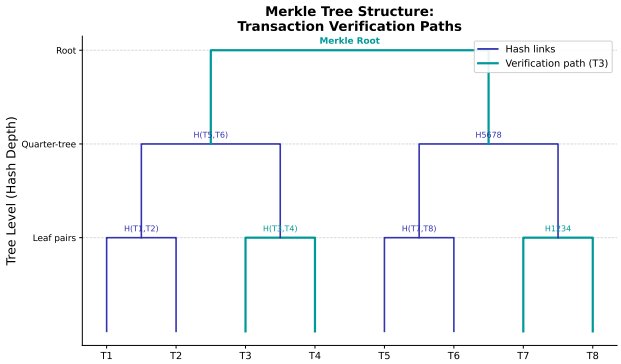
$$\Pr\big[\text{bit}_i(H(m)) \neq \text{bit}_i(H(m'))\big] \approx 0.5$$

Chain integrity: Each block stores $H(\text{block}_{k-1})$. Changing block k invalidates all subsequent hashes. Cost of rewriting n blocks:

$$C_{\text{rewrite}} = O(2^{n \cdot d}) \quad \text{where } d = \text{difficulty bits}$$

Three properties, one guarantee: you cannot reverse, duplicate, or forge a hash. This is why changing one block breaks the entire chain.

How Does a Merkle Tree Let You Verify One Transaction Without Downloading the Whole Chain?



Merkle proof. To verify transaction T_k in a tree of n leaves, a light client needs only $\lceil \log_2 n \rceil$ sibling hashes:

- $n = 8$ transactions $\Rightarrow$ 3 hashes suffice
- Full node sends proof path; client re-hashes upward
- If recomputed root = block header root, T_k is confirmed

SPV (Simple Payment Verification):

- Download block headers only ($\sim$80 bytes each)
- Request Merkle proof for your transaction
- Verify $O(\log n)$ vs. downloading $O(n)$

Merkle trees turn O(n) verification into O(log n). This is why a phone can verify a Bitcoin transaction without downloading the entire blockchain.

Can You Build a Working Hash Chain in 20 Lines of Python?

```python
import hashlib, dataclasses, json

@dataclasses.dataclass
class Block:
    index: int
    data: str
    prev_hash: str = ""
    def compute_hash(self):
        content = json.dumps({"i": self.index, "d": self.data,
                              "p": self.prev_hash}, sort_keys=True)
        return hashlib.sha256(content.encode()).hexdigest()

def build_chain(records):
    chain = []
    for i, data in enumerate(records):
        prev = chain[-1].compute_hash() if chain else "0" * 64
        chain.append(Block(index=i, data=data, prev_hash=prev))
    return chain

chain = build_chain(["Genesis", "Alice->Bob: 5 BTC", "Bob->Carol: 2 BTC"])
# Tamper with block 1 and watch the avalanche:
chain[1].data = "Alice->Bob: 500 BTC" # every subsequent hash now differs
for b in chain:
    print(f"Block {b.index}: {b.compute_hash()[:16]}... prev={b.prev_hash[:16]}...")
```

Twenty lines of Python to build an immutable chain. Change one character in block 2 and every subsequent hash breaks – that is the avalanche effect in action.

What Makes Proof of Work a Game – and What Is the Nash Equilibrium?

Mining as a Poisson game. Probability of finding a valid block:

$$P(\text{block found}) = 1 - \left(1 - 2^{-d}\right)^{n \cdot h}$$

where d = difficulty bits, n = nonce attempts, h = hash rate.

Nash equilibrium analysis:

$$\begin{aligned} \text{Honest payoff:} \quad & \pi_H = p \cdot R - C_e \\ \text{Attack payoff:} \quad & \pi_A = p' \cdot R' - C'_e \quad \text{where } p' < p \text{ for hashrate} < 0.5 \end{aligned}$$

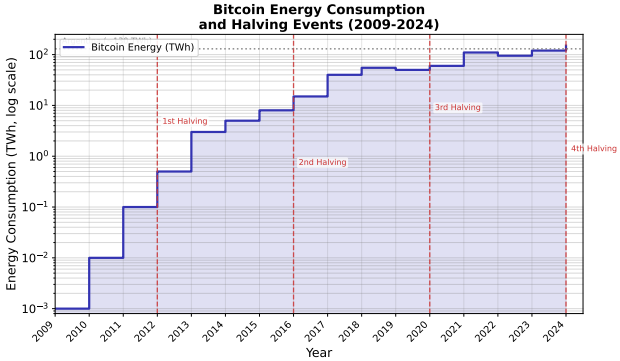
Incentive compatibility:

$$\pi_H > \pi_A \iff \text{attacker hashrate fraction} < 0.5$$

51% attack cost: An attacker needs $> \frac{1}{2} \sum_i h_i$ total hashrate. At current Bitcoin network ($\sim$600 EH/s), the hardware and electricity cost exceeds \$10B.

Mining is a game where honesty pays better than cheating – as long as no single player controls more than half the hashrate. That threshold is the Nash equilibrium boundary.

How Much Energy Does Trust Cost – and When Did the Price Change?



Halving events cut the block reward in half every 210,000 blocks (∼4 years):

- 2012: 50 → 25 BTC
- 2016: 25 → 12.5 BTC
- 2020: 12.5 → 6.25 BTC
- 2024: 6.25 → 3.125 BTC

Energy follows incentives:

- Reward ↓ ⇒ marginal miners exit
- Price ↑ ⇒ new miners enter
- Net energy tracks Price × Reward
- Bitcoin now rivals Argentina's annual consumption

Every halving cuts the mining reward in half. Energy consumption follows price follows reward – the step pattern reveals Bitcoin's built-in deflationary heartbeat.

Can You Simulate a Proof-of-Work Mining Race?

```python
import hashlib, time, random

def mine_block(data, difficulty):
    """Find nonce such that hash starts with `difficulty` zero bits."""
    target = "0" * difficulty
    nonce = 0
    while True:
        text = f"{data}{nonce}"
        h = hashlib.sha256(text.encode()).hexdigest()
        if h[:difficulty] == target:
            return nonce, h
        nonce += 1

# Demonstrate difficulty scaling
for d in range(1, 6):
    start = time.time()
    nonce, h = mine_block("Block #42 data", d)
    elapsed = time.time() - start
    print(f"Difficulty {d}: nonce={nonce:>8,} time={elapsed:.4f}s"
          f"  hash={h[:20]}...")

# Key insight: doubling difficulty roughly doubles average attempts
# Bitcoin adjusts difficulty every 2016 blocks to maintain ~10 min
```

Run this code and watch: double the difficulty, double the average nonce attempts. The difficulty adjustment is what keeps Bitcoin producing one block every 10 minutes regardless of total hashrate.

When Does Staking Your Own Money Make You More Honest Than Spending Electricity?

Validator expected return:

$$E[R] = \frac{S_i}{\sum_j S_j} \cdot R_{\text{block}} - C_{\text{opportunity}}$$

where S_i = validator's stake, R_{block} = block reward.

Slashing penalty: $P_{\text{slash}} = \alpha \cdot S_i$ where $\alpha \in [0, 1]$ is severity.
Incentive condition:

$$E[R_{\text{honest}}] > E[R_{\text{attack}}] - P_{\text{slash}}$$

Nothing-at-stake problem: In PoS, the cost of validating on multiple competing chains is $C_{\text{multichain}} \approx 0$ (no electricity cost). Solution: slashing condition

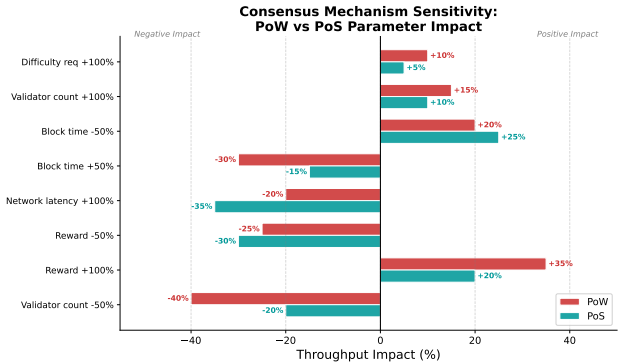
$$P_{\text{slash}} > E[R_{\text{multichain}}]$$

PoW vs. PoS security model:

- PoW security = *flow* cost (electricity per block)
- PoS security = *stock* cost (capital locked)

PoW burns electricity to prove honesty. PoS locks capital and threatens to slash it. Both create skin in the game – the question is whether you prefer to burn energy or risk wealth.

How Sensitive Are Consensus Mechanisms to Parameter Changes?



Tornado chart shows how security/performance changes when each parameter shifts by ±50–100%:

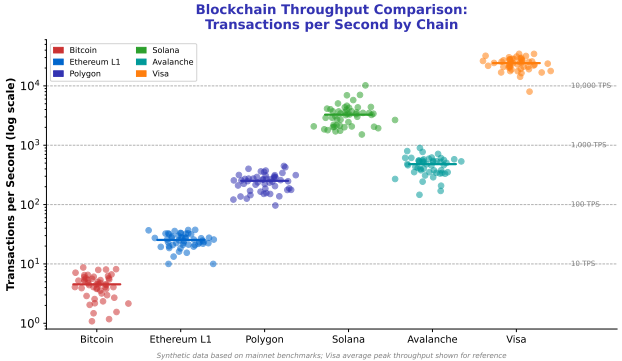
- PoW is most sensitive to *validator count reduction* (−40% security)
- PoS is most sensitive to *network latency* (−35%)
- Reward changes hit both, but PoW recovers faster

Design implication:

- PoW → protect miner diversity
- PoS → minimize network latency
- Both → reward must exceed opportunity cost

The tornado chart reveals which knobs matter most. Reducing validator count degrades PoW security far more than PoS – a key reason regulators prefer Proof of Stake.

Which Blockchain Can Actually Handle Visa-Scale Transactions?



Beeswarm plot: Each dot is one observed TPS measurement. Spread shows real-world variance.

- Bitcoin: ∼5 TPS (by design)
- Ethereum L1: ∼25 TPS
- Polygon / Avalanche: 250–450 TPS
- Solana: high variance, 1,000–5,000 observed
- Visa: ∼24,000 sustained

Why throughput alone misleads:

- High TPS often sacrifices decentralization
- Finality ≠ confirmation speed
- Theoretical max ≠ sustained throughput

Solana claims 65,000 TPS but Visa processes 24,000 sustained. The gap between theoretical maximum and observed throughput is where marketing meets reality.

Why Can You Only Have Two of Three – and Can You Prove It?

Model. Three normalized properties $D, S, T \in [0, 1]$:

- D = decentralization (number of independent validators)
- S = security (cost to attack consensus)
- T = throughput / scalability (transactions per second)

Buterin's conjecture: $D + S + T \leq 2$

Communication complexity argument:

$$\text{Throughput } T \Rightarrow O(n \cdot T) \text{ messages among } n \text{ validators}$$

$$\text{BFT Security } S \Rightarrow O(n^2) \text{ messages for agreement}$$

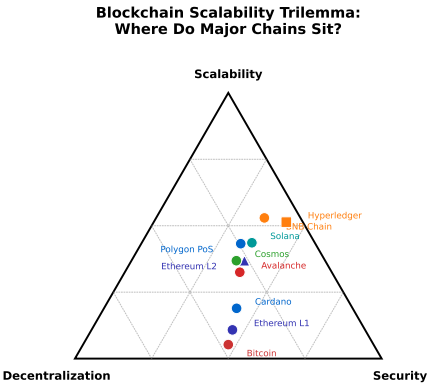
$$\text{Full Decentralization } D \Rightarrow \text{all } n \text{ nodes process all transactions}$$

Combining all three requires $O(n^2 \cdot T)$ – infeasible at scale.

Layer 2 relaxation: L2 achieves $T' > T$ by reducing effective n (validator subset), thus lowering D locally while inheriting S from L1.

The trilemma is not a law of physics – it is a communication complexity bound. Layer 2 solutions do not break it; they shift the trade-off to a different layer.

Where Do Major Blockchains Sit on the Trilemma Triangle?



Reading the ternary plot: Proximity to a vertex means higher score on that dimension.

- **Bitcoin:** top-left edge (high D + S, low T)
- **Solana:** bottom-right edge (high S + T, low D)
- **Hyperledger:** near Scalability vertex (enterprise, permissioned)
- **Ethereum L2s:** interior – partial on all three

Key insight: No chain sits in the center. Every blockchain is a values statement about which property to sacrifice.

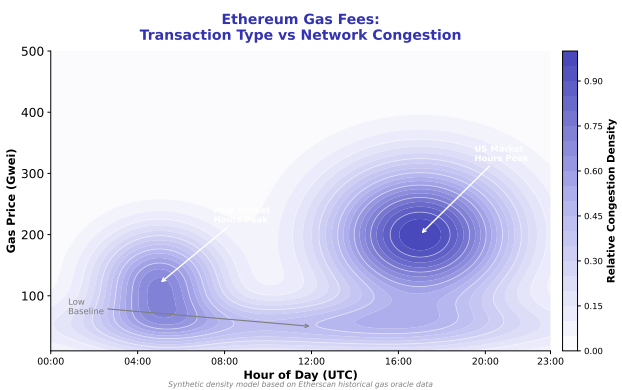
No blockchain sits in the center of the triangle. Every chain is a values statement: Bitcoin chose decentralization and security; Solana chose security and scalability.

Can You Build a Merkle Tree and Verify a Transaction Proof?

```python
import hashlib

def sha256(data):
    return hashlib.sha256(data.encode()).hexdigest()

def build_merkle_tree(txns):
    """Build tree bottom-up; return list of levels."""
    layer = [sha256(tx) for tx in txns]
    tree = [layer]
    while len(layer) > 1:
        if len(layer) % 2:       # odd count: duplicate last
            layer.append(layer[-1])
        layer = [sha256(layer[i] + layer[i+1])
                 for i in range(0, len(layer), 2)]
        tree.append(layer)
    return tree # tree[-1][0] is the Merkle root

def get_proof(tree, idx):
    """Return sibling hashes needed to verify leaf `idx`."""
    proof = []
    for level in tree[:-1]:
        sib = idx ^ 1 # toggle last bit -> sibling index
        proof.append(("L" if idx % 2 else "R", level[min(sib, len(level)-1)]))
        idx //= 2
    return proof # O(log n) hashes

txns = [f"tx_{i}" for i in range(8)]
tree = build_merkle_tree(txns)
print(f"Root: {tree[-1][0][:16]}... Proof for tx_2: {len(get_proof(tree,2))} hashes")
```

This is how your phone verifies a Bitcoin payment: instead of downloading 500 GB of blockchain, it checks 3 hashes. Merkle trees make light clients possible.

How Much Does It Cost to Use Ethereum – and What Drives the Price?



Contour density shows transaction volume by hour (UTC) and gas price (gwei).

- Peak: US trading hours (14:00–20:00 UTC)
- Secondary peak: Asian session (02:00–08:00)
- Off-peak: $<$20 gwei; peak: 200$+$ gwei

EIP-1559 fee mechanism:

- **Base fee:** algorithmically adjusted, *burned*
- **Priority fee (tip):** goes to validator
- Under high demand, ETH becomes *deflationary* (burn $>$ issuance)

Gas fees are the price of scarcity. When the network is congested, users bid against each other for block space. EIP-1559 burns the base fee, making ETH deflationary under high demand.

How Do You Formally Model a Smart Contract as a State Machine?

Definition. A smart contract is a deterministic FSM $(Q, \Sigma, \delta, q_0, F)$:

- Q = set of states (contract storage configurations)
- Σ = set of inputs (function calls with parameters)
- $\delta : Q \times \Sigma \to Q$ = transition function (deterministic execution)
- q_0 = initial state (deployment)
- F = final states (most contracts: $F = \emptyset$, run indefinitely)

Reentrancy as state violation. δ must be atomic, but an external call interrupts the transition – re-entry occurs before state update:

$$q \xrightarrow{\sigma_1 \text{ (partial)}} q^* \xrightarrow{\sigma_1 \text{ (re-enter)}} q^{**} \neq \delta(q, \sigma_1)$$

Formal verification goal: Prove for all reachable states and inputs:

$$\forall\, s \in Q,\ \forall\, \sigma \in \Sigma : \quad \text{invariant}\big(\delta(s, \sigma)\big) \quad \text{e.g., } \sum \text{balances} = \text{total\_supply}$$

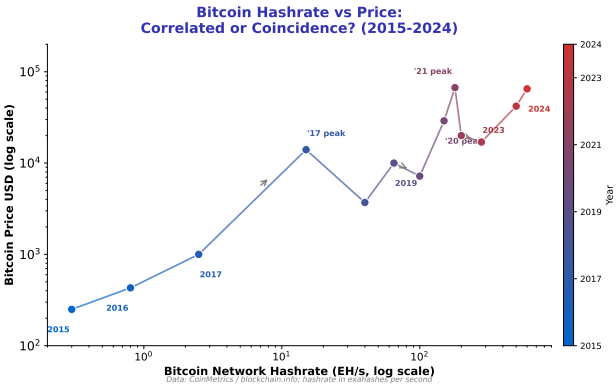
A smart contract is a state machine. Reentrancy attacks exploit non-atomic transitions – the state changes mid-step. Formal verification catches these by proving invariants hold across ALL transitions.

Can You Simulate a Simple Smart Contract in Python?

```python
class TokenVault:
    """Vulnerable: sends ETH before updating balance."""
    def __init__(self):
        self.balances = {}
    def deposit(self, user, amount):
        self.balances[user] = self.balances.get(user, 0) + amount
    def withdraw(self, user):
        amt = self.balances.get(user, 0)
        self._send(user, amt)       # external call FIRST (bug!)
        self.balances[user] = 0     # state update SECOND

    def _send(self, user, amt):     # simulates external call
        if hasattr(user, "on_receive"):
            user.on_receive(self, amt) # attacker re-enters here

class Attacker:
    def __init__(self): self.loot = 0
    def on_receive(self, vault, amt):
        self.loot += amt
        if self.loot < amt * 3:     # re-enter up to 3x
            vault.withdraw(self)

class FixedVault(TokenVault):
    """Checks-effects-interactions: update state BEFORE send."""
    def withdraw(self, user):
        amt = self.balances.get(user, 0)
        self.balances[user] = 0     # state update FIRST (fix!)
        self._send(user, amt)       # external call SECOND
```

The fix is one line: update the balance BEFORE sending. The checks-effects-interactions pattern is the single most important security rule in smart contract development.

Does Bitcoin Hashrate Follow Price – or Does Price Follow Hashrate?



Connected scatter: Each dot is one month (2015–2024), connected chronologically.

- Color gradient: early (blue) to recent (red)
- Both axes on log scale
- The path spirals upper-right: both price and hashrate trend upward

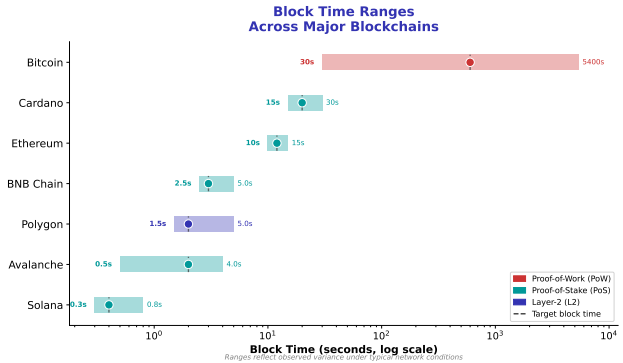
Feedback loop:

1. Price ↑ ⇒ mining more profitable
2. Miners enter ⇒ hashrate ↑
3. Hashrate ↑ ⇒ security ↑
4. Security ↑ ⇒ confidence ↑ ⇒ price ↑

Granger causality: Price leads hashrate by 3–6 months.

Price attracts miners. Miners increase hashrate. Hashrate increases security. Security attracts capital. The connected scatter traces this feedback loop across a decade.

How Long Do You Actually Wait for a Block?



Span chart: Each bar spans from minimum to maximum observed block time. Dot marks median.

- **Bitcoin:** 30s – 90 min (target 10 min)
- **Ethereum:** 10 – 15s (target 12s)
- **Solana:** 0.3 – 0.8s (target 0.4s)
- Variance matters for user experience

Why the tail matters:

- "Average 10 minutes" hides a 90-minute worst case
- Finality = block time $\times$ confirmations
- For payments: median matters; for settlement: worst-case matters

Bitcoin's "average 10 minutes" hides a range from 30 seconds to 90 minutes. Block time is a distribution, not a number – and the tail matters for user experience.

How Do Zero-Knowledge Proofs Work – and What Do They Actually Prove?

Definition. An interactive proof system (P, V) for language L satisfies:

$$\begin{aligned} \textbf{Completeness:} \quad & x \in L \implies \Pr[V \text{ accepts}] \geq 1 - \epsilon \\ \textbf{Soundness:} \quad & x \notin L \implies \Pr[V \text{ accepts}] \leq \epsilon \\ \textbf{Zero-knowledge:} \quad & \exists \text{ simulator } S \text{ s.t. } \text{View}_V(P \leftrightarrow V) \approx S(x) \end{aligned}$$

Schnorr protocol (prove knowledge of $x = \log_g h$ without revealing x):

1. Prover sends commitment $a = g^r$ (random r)
2. Verifier sends challenge c
3. Prover sends response $z = r + cx$
4. Verifier checks $g^z = a \cdot h^c$

Application: ZK-rollups batch n transactions into one proof, verified on L1 in $O(1)$.

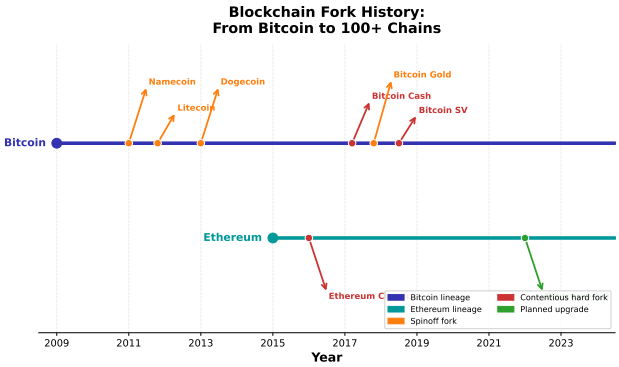
Zero-knowledge proofs let you prove you know a secret without revealing it. This is not a trick – it is a mathematical guarantee with three formal properties: completeness, soundness, and zero-knowledge.

Can You Verify a Zero-Knowledge Proof Without Understanding the Math?

```python
import random

# Simplified Schnorr-like ZKP over integers (educational)
p = 104729        # prime modulus
g = 2             # generator
secret = 42       # prover's secret (never revealed!)
h = pow(g, secret, p) # public value: h = g^secret mod p

def prover_commit():
    r = random.randint(1, p - 2)
    a = pow(g, r, p)          # commitment
    return r, a

def prover_respond(r, c):
    return (r + c * secret) % (p - 1) # z = r + c*x

def verifier_check(a, c, z, h):
    lhs = pow(g, z, p)        # g^z
    rhs = (a * pow(h, c, p)) % p # a * h^c
    return lhs == rhs

# Run the protocol
r, a = prover_commit()
c = random.randint(1, p - 2) # verifier challenge
z = prover_respond(r, c)
ok = verifier_check(a, c, z, h)
print(f"Proof valid: {ok} | Secret revealed? Never. h={h}")
```

The verifier never sees the secret, yet becomes convinced the prover knows it. Run this code: the proof verifies every time, but the secret never appears in the output.

How Did Bitcoin Become 100 Blockchains?



Fork types:

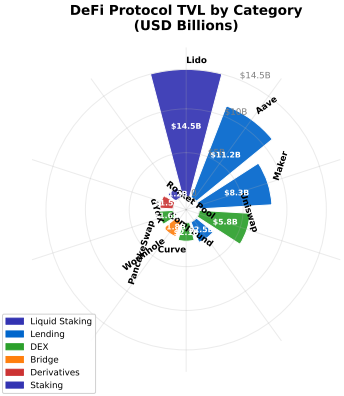
- Contentious hard fork: community splits over rules (Bitcoin Cash 2017, Ethereum Classic 2016)
- Planned upgrade: coordinated change (Ethereum Merge 2022)
- Spinoff: code copied, new chain (Litecoin 2011, Dogecoin 2013)

Why forks happen:

- No central authority to resolve disputes
- "Disagreement = divergence" in decentralized governance
- Forks are exit rights made manifest
- Most forks lose network effect and fade

Every fork is a governance failure that becomes a feature. Bitcoin spawned 100+ chains because decentralized systems have no appeals court – disagreement means divergence.

Where Is the Money in DeFi – and How Concentrated Is It?



Radial bar chart: Bar length $\propto$ total value locked.

- **Lido** (liquid staking): **$**14.5B
- **Aave** (lending): **$**11.2B
- **Maker** (lending): **$**8.3B
- **Uniswap** (DEX): **$**5.8B
- Top 5 hold $>$ 60% of all DeFi TVL

Concentration risk:

- A single exploit in Lido or Aave affects $>$ 20% of DeFi
- Composability = contagion vector
- "Decentralized" finance is more concentrated than traditional banking's top 5

DeFi promises decentralization but the money tells a different story. The top 5 protocols hold over 60 percent of all value locked – a single exploit would be systemic.

What Is the Formal Security Bound of a 51% Attack?

Nakamoto's random walk model. Attacker with hashrate fraction $q < 0.5$ tries to build a longer chain than the honest majority ($p = 1 - q$).

Probability of catching up after z confirmations:

$$P(\text{catch up} \mid z, q) = \left(\frac{q}{p}\right)^z \quad \text{for } q < 0.5$$

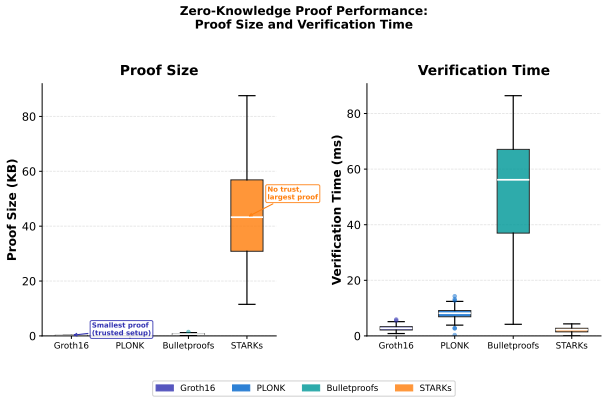
Numerical examples:

Attacker q	$z = 1$	$z = 3$	$z = 6$	$z = 12$
$q = 0.10$	11.1%	0.14%	<0.001%	~ 0
$q = 0.25$	33.3%	3.70%	0.14%	<0.001%
$q = 0.30$	42.9%	7.89%	0.62%	0.004%
$q = 0.45$	81.8%	54.8%	30.1%	9.05%

Cost of attack: $C_{\text{attack}} = R_{\text{block}} \cdot n_{\text{blocks}} \cdot (1 - q/p)$ – the opportunity cost of *not* mining honestly.

Six confirmations reduce attack probability to 0.024 percent for an attacker with 30 percent hashrate. This is why Bitcoin wallets wait for 6 blocks – the math guarantees exponential security decay.

How Fast and Small Can a Zero-Knowledge Proof Get?



Box plot: Four ZK systems compared on proof size (KB) and verification time (ms).

- **Groth16:** smallest proofs (∼0.2 KB) but requires trusted setup ceremony
- **PLONK:** universal setup, moderate size
- **Bulletproofs:** no setup, but slow verification (∼50 ms)
- **STARKs:** no trusted setup, largest proofs (∼45 KB), fastest verification

Trade-off:

- Trusted setup ↔ proof size
- Proof size ↔ verification time
- No system dominates on all axes

Groth16 proofs are tiny (200 bytes) but require a trusted setup ceremony. STARKs need no trust but produce 45 KB proofs. The ZK design space is a trade-off, not a solution.

What Have We Learned – and What Remains Unsolved?

Five sections, one unifying theme: the scalability trilemma is not a bug – it is a values statement.

- **Cryptographic Foundations:** Hash functions provide immutability; Merkle trees enable efficient verification. The math is settled – $O(\log n)$ is optimal.
- **Consensus Game Theory:** PoW and PoS are incentive-compatible games with different cost structures. The Nash equilibrium holds at $< 50\%$ attacker hashrate/stake. The open question: can we reduce energy cost without reducing security?
- **Scalability Trilemma:** Communication complexity bounds force trade-offs. Layer 2 shifts the bound, not breaks it. No blockchain occupies the center of the ternary plot.
- **Smart Contracts:** Formally, they are finite state machines. Reentrancy exploits non-atomic transitions. Formal verification is possible but not yet standard practice.
- **Advanced Topics:** ZKPs enable privacy and scalability but introduce proof-size/verification trade-offs. DeFi is composable but concentrated. Forks are governance made visible.

What remains unsolved: quantum resistance, MEV (miner extractable value), cross-chain interoperability, and regulatory classification of DeFi tokens.

Five sections, one lesson: blockchain is not a technology problem – it is a values problem. Every design choice trades one property for another. The trilemma is the unifying framework.

Key Takeaways

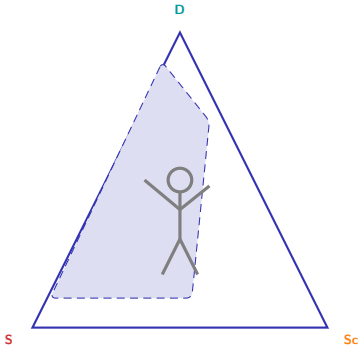
1. **Hash functions are the foundation:** Pre-image resistance, collision resistance, and the avalanche effect make blockchain immutability mathematically inevitable – changing one block breaks every subsequent hash.
2. **Consensus is a game:** PoW and PoS create incentive structures where honest behavior is the Nash equilibrium – as long as no single actor controls $> 50\%$ of hashrate or stake.
3. **The trilemma is a constraint, not a choice:** Communication complexity limits force every blockchain to sacrifice at least one of decentralization, security, or scalability. Layer 2 shifts the trade-off but does not eliminate it.
4. **Smart contracts are state machines:** Reentrancy attacks exploit non-atomic state transitions. The checks-effects-interactions pattern is the single most important defense.
5. **Zero-knowledge proofs enable "trust without disclosure":** Completeness, soundness, and zero-knowledge are formally provable properties – but proof systems trade off size, speed, and trusted setup.
6. **Blockchain design is a values statement:** Every protocol choice reveals a priority. Bitcoin chose decentralization; Solana chose scalability; Ethereum is trying to have it both ways via Layer 2.

Six takeaways, one framework. If you remember one thing: the scalability trilemma means every blockchain is a values statement disguised as a technology choice.

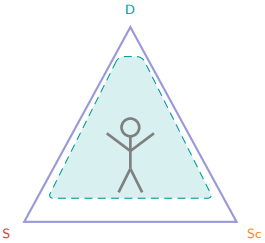
References and Further Reading

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. `https://bitcoin.org/bitcoin.pdf`
- Buterin, V. (2014). *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. Ethereum White Paper.
- Goldwasser, S., Micali, S., & Rackoff, C. (1989). The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1), 186–208.
- Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 382–401.
- Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., & Felten, E. W. (2015). SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. *IEEE S&P*, 104–121.
- Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A Survey of Attacks on Ethereum Smart Contracts (SoK). *POST 2017*, LNCS 10204, 164–186.

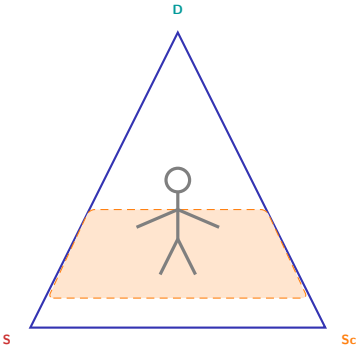
Start with Nakamoto (2008) for the vision, Buterin (2014) for the programmable extension, and Bonneau et al. (2015) for the honest security analysis.



Bitcoin's choice



Layer 2's promise: stretch, don't choose



Enterprise's choice

You cannot cover three corners with a two-corner blanket. But you can build a second layer.