CS 521 — Foundations of Crypto

Round 1 Presentations — Overview

Spring 2026 · Prof. Grigore Rosu · UIUC

Merkle Patricia Tries

Team: Mike Cheng & Howard Hsu

Merkle Patricia Trie data structure, role in Ethereum state management, comparison with simple Merkle trees and other authenticated data structures.

What the team presented

- **Why Ethereum needs MPT** — simple Merkle trees scale poorly; MPT combines hashing with a hexary prefix trie
- **Three node types** — extension, branch (16 children + optional value), and leaf
- **Two-nibble flag encoding** — disambiguates extension vs. leaf and handles odd-length paths
- **Four Ethereum tries** — state, storage, transaction, and receipt tries — with RLP encoding and bloom filters
- **Comparisons** — MPT vs. simple Merkle (depth, lookup); preview of Verkle tree proof-size savings

What the team will build

Implement a Merkle Patricia Trie supporting insert, lookup, delete, and proof generation/verification. Demonstrate state root computation and efficient proof-of-inclusion.

Challenge: Stretch: include deletion + state-root recomputation, and benchmark proof sizes vs. a plain Merkle tree on the same data.

Mike Cheng & Howard Hsu — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Merkle Patricia Tries

Team: Wilson Yu & Asher Sun

Merkle Patricia Trie data structure, role in Ethereum state management, comparison with simple Merkle trees and other authenticated data structures.

What the team presented

- **MPT decomposed** — Merkle = hashed tree; Patricia = prefix-compression trie; combined into a secure trie
- **Three Ethereum tries walked through** — state, storage, receipt — and transaction trie for block-local proofs
- **Three node types** — branch (17-element array), extension (2-element), leaf (2-element)
- **Light-client proofs** — how a Merkle path in the transaction trie powers cheap payment verification
- **Future comparison** — MPT O(log_16 N) vs. accumulators' constant-size proofs and trusted setup tradeoff

What the team will build

Implement a Merkle Patricia Trie supporting insert, lookup, delete, and proof generation/verification. Demonstrate state root computation and efficient proof-of-inclusion.

Challenge: Stretch: demonstrate a light-client verification flow where a proof for a single account works without any other state data.

Wilson Yu & Asher Sun — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Automated Market Makers (AMMs)

Team: Daniel S Li & Kavin Sankar

AMM design: constant product formula (Uniswap v2), concentrated liquidity (Uniswap v3), impermanent loss, sandwich attacks on AMMs, and comparisons with order-book exchanges.

(Team presented AMMs instead of their assigned zk-SNARKs topic.)

#topic-02a-zk-snarks

What the team presented

- **Why AMMs replace CLOBs on-chain** — need always-on liquidity, no intermediary, smart-contract compatible
- **Uniswap v2 math** — X·Y = K; instantaneous price; 0.3% LP fee
- **Price impact + slippage** — demonstrated 10 ETH / 20k USDC pool and how large trades move the curve
- **Impermanent loss** — formal intuition: LPs automatically sell winners and buy losers
- **Uniswap v3 concentrated liquidity** — LP-as-NFT; 10–100× capital efficiency within chosen range
- **MEV + sandwich attacks** — mempool visibility + deterministic pricing create the exploit

What the team will build

Implement a simplified AMM smart contract (constant product) with swap, add liquidity, and remove liquidity functions. Deploy on a testnet and demonstrate price impact and impermanent loss scenarios.

Challenge: Stretch: reproduce a sandwich attack on your deployed AMM and measure the victim's extra slippage cost.

Daniel S Li & Kavin Sankar — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

zk-STARKs

Team: Aditya Ballabh & Steven Kusuman

Zero-knowledge proof fundamentals with focus on zk-STARKs: transparency (no trusted setup), the FRI protocol, hash-based construction, and post-quantum security properties. Compare with zk-SNARKs on proof size, verification time, and trust assumptions.

What the team presented

- **ZK proof foundations** — completeness, soundness, zero-knowledge; Hamiltonian cycle interactive example
- **Sigma protocols → Fiat-Shamir** — how non-interactivity is obtained via a random oracle hash
- **STARK transparency** — no trusted setup; hash-based commitments vs. SNARK CRS / toxic waste
- **Arithmetization** — AIR execution trace; Merkle-committed rows; boundary + transition polynomial constraints
- **FRI protocol** — recursive folding to prove low-degree of committed polynomial in log rounds
- **SNARK vs. STARK comparison** — proof size, verifier time, prover time, post-quantum security

What the team will build

Implement a simplified STARK-like proof system for a simple computation (e.g., Fibonacci verification), or use an existing STARK library to prove and verify a non-trivial computation.

Challenge: Stretch: produce a concrete benchmark table (proof size, prover time, verifier time) for your STARK vs. a reference SNARK on the same computation.

Aditya Ballabh & Steven Kusuman — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Hash-Based Accumulators and Authenticated Data Structures

Team: Saman Dehghan Tooranposhti & Edward Li

RSA accumulators, Merkle accumulators, and vector commitments. Use in stateless blockchain clients, certificate transparency, and revocation. Compare efficiency and trust assumptions.

What the team presented

- **ADS model** — untrusted prover + polynomial-time verifier; zero-trust client/server interaction
- **RSA accumulators** — elements mapped to primes; $G^{\prod p_i} \bmod N$; constant-size membership witness
- **Merkle trees** — log N proofs; collision-resistance only; no trusted setup
- **Vector commitments** — position-binding; KZG, Verkle, lattice schemes for sub-vector openings
- **Blockchain applications** — stateless clients; certificate transparency; certificate revocation
- **Efficiency table** — RSA vs. Merkle vs. Verkle on accumulator size, proof size, update cost

What the team will build

Implement and benchmark multiple accumulator schemes (Merkle-based and at least one alternative) with membership and non-membership proof generation and verification.

Challenge: Stretch: benchmark proof-size and verification-time against a trillion-element workload to stress the asymptotics.

Saman Dehghan Tooranposhti & Edward Li — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Hash-Based Accumulators and Authenticated Data Structures

Team: Rudy Juarez-Ovallos & Sanju Golla

RSA accumulators, Merkle accumulators, and vector commitments. Use in stateless blockchain clients, certificate transparency, and revocation. Compare efficiency and trust assumptions.

What the team presented

- **Accumulator motivation** — digest replaces full dataset; stateless-client storage savings
- **Merkle trees** — root hash; log-N membership proofs; worked example with 4-account tree
- **RSA accumulators** — modular exponentiation on primes; constant-size membership proofs
- **RSA worked example** — G=2, primes {3,5,7,11} → accumulator value 32, witness verification
- **Vector commitments** — position-binding; commitment size independent of vector length
- **Tradeoffs per use case** — Merkle for certificate transparency; RSA for revocation (non-membership)

What the team will build

Implement and benchmark multiple accumulator schemes (Merkle-based and at least one alternative) with membership and non-membership proof generation and verification.

Challenge: Stretch: include a benchmark of insert/update bandwidth where witnesses need refreshing on each change.

Rudy Juarez-Ovallos & Sanju Golla — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Bitcoin Script and Transaction Types

Team: Yueqiang Wu & Chenxin Yan

Bitcoin's scripting language: design philosophy (intentionally not Turing-complete), standard transaction types (P2PKH, P2SH, P2WPKH, P2WSH, multisig), Taproot/Schnorr upgrades, and security properties.

What the team presented

- **Why Bitcoin has a script language** — UTXO model requires programmable spend conditions; Forth-inspired stack machine
- **Design philosophy** — non-Turing-complete by choice; limited opcodes; predictable validation time
- **Locking/unlocking mechanics** — scriptSig + scriptPubKey; step-by-step stack execution
- **Standard transaction types** — P2PK → P2PKH → P2SH → P2WPKH/P2WSH (SegWit)
- **SegWit benefits** — separates signatures; fixes malleability; increases effective block capacity
- **ECDSA → Schnorr → Taproot** — linearity; 64-byte fixed sigs; batch verification; MAST hides unused branches

What the team will build

Build a Bitcoin Script interpreter that parses and executes standard scripts on a stack machine. Demonstrate correct validation of multiple transaction types.

Challenge: Stretch: show your interpreter correctly executing a P2SH-wrapped multisig and a Taproot keypath spend.

Yueqiang Wu & Chenxin Yan — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Bitcoin Script and Transaction Types

Team: Lawrence Wang & Jeff Zou

Bitcoin's scripting language: design philosophy (intentionally not Turing-complete), standard transaction types (P2PKH, P2SH, P2WPKH, P2WSH, multisig), Taproot/Schnorr upgrades, and security properties.

What the team presented

- **Transaction anatomy** — inputs/outputs, TXID pointers, locking/unlocking scripts
- **Turing-incomplete by design** — solves the halting problem; hard limits on stack, script size, opcodes
- **Stack execution walkthrough** — DUP, HASH160, EQUALVERIFY, CHECKSIG step-by-step
- **P2PK → P2PKH → P2WPKH** — privacy and fee evolution; witness separation saves 75% on signature data
- **P2MS vs. P2SH** — native multisig limits (3 keys); P2SH raises limit to 15; OP_0 dummy workaround
- **OP_RETURN** — provably unspendable, prunable data carrier; 80-byte limit
- **SegWit + Schnorr + Taproot** — malleability fix; key aggregation; key-path vs. script-path indistinguishability

What the team will build

Build a Bitcoin Script interpreter that parses and executes standard scripts on a stack machine. Demonstrate correct validation of multiple transaction types.

Challenge: Stretch: demonstrate validation of a non-standard script being rejected as a negative-case test.

Lawrence Wang & Jeff Zou — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Bitcoin Payment Channels and the Lightning Network

Team: Shengxiang Qi & Yuqing Zhang

Off-chain scaling for Bitcoin via payment channels: funding transactions, commitment transactions, HTLCs, onion routing, and the Lightning Network architecture. Discuss capacity, routing challenges, and privacy properties.

What the team presented

- **Bitcoin scaling trilemma** — 3–7 TPS base-layer bottleneck; why L2 is the lever
- **Channel lifecycle** — open (2-of-2 funding), update (signed commitment tx, not broadcast), close
- **Revocation + penalty** — older state becomes revocable; honest party can sweep funds if cheating
- **Multi-hop via HTLCs** — hashlock + timelock; worked example Alice → Bob → Carol with secret propagation
- **Routing challenges** — per-channel capacity, dynamic liquidity, offline nodes
- **Privacy via onion routing** — per-hop layer decryption hides full route; on-chain open/close still visible

What the team will build

Implement a simplified two-party payment channel with state updates and dispute resolution using Bitcoin Script-like semantics on a local testnet or simulation.

Challenge: Stretch: simulate a cheat-and-penalty scenario end-to-end and measure how the delay window affects safety.

Shengxiang Qi & Yuqing Zhang — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Bitcoin Payment Channels and the Lightning Network

Team: Jingzhi Bao & Bofan Chen

Off-chain scaling for Bitcoin via payment channels: funding transactions, commitment transactions, HTLCs, onion routing, and the Lightning Network architecture. Discuss capacity, routing challenges, and privacy properties.

What the team presented

- **Scaling gap** — Bitcoin base layer ≈7–10 TPS vs. daily-payment needs
- **Layer-2 paradigm** — lock on-chain, transact off-chain, settle net balances back
- **Funding via 2-of-2 multisig UTXO** — a shared vault whose internal state is never written on-chain
- **Asymmetric commitment transactions** — broadcaster pays a delay; counterparty can react
- **Revocation secrets** — exchanging them invalidates prior states; penalty transactions punish cheaters
- **Multi-hop + HTLCs** — hashlock + timelock give atomic success-or-fail semantics
- **Onion routing** — each hop sees only predecessor and successor; route + amount hidden

What the team will build

Implement a simplified two-party payment channel with state updates and dispute resolution using Bitcoin Script-like semantics on a local testnet or simulation.

Challenge: Stretch: demonstrate the penalty-transaction mechanic under a simulated cheat attempt on your testnet.

Jingzhi Bao & Bofan Chen — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Bitcoin Mining and Selfish Mining

Team: Yuhang Yang & Jiahua Zhao

Mining economics: hash rate, difficulty adjustment, pool strategies (PPLNS, PPS), energy consumption, and the selfish mining attack (Eyal & Sirer, 2014). Analyze the threshold at which selfish mining becomes profitable.

What the team presented

- **PoW refresher** — difficulty target, longest-chain rule, coinbase + fees
- **Stale blocks & forks** — propagation delay creates the opportunity for strategic withholding
- **Pool payouts** — PPS (pool absorbs variance) vs. PPLNS (miner absorbs variance)
- **Difficulty retargeting** — every 2016 blocks; energy rises as price rises, throughput stays flat
- **Selfish-mining strategy** — withhold block, fork-race on L=1, knockout on L≥2
- **Threshold formula** — $\alpha^* = (1-\gamma)/(3-2\gamma)$; $\alpha^* \approx 33\%$ at $\gamma=0$ but drops to ~25% at $\gamma=0.5$
- **Centralization snowball** — more profitable pool → better propagation → higher γ → lower threshold

What the team will build

Build a mining simulator with configurable parameters (hash rate distribution, network latency, pool size) that reproduces the selfish mining advantage threshold and visualizes results.

Challenge: Stretch: visualize the α threshold as γ sweeps 0→1 and overlay empirical results from your simulator.*

Yuhang Yang & Jiahua Zhao — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Bitcoin Mining and Selfish Mining

Team: Wenyi Xie & Rui Yu

Mining economics: hash rate, difficulty adjustment, pool strategies (PPLNS, PPS), energy consumption, and the selfish mining attack (Eyal & Sirer, 2014). Analyze the threshold at which selfish mining becomes profitable.

What the team presented

- **Hash-rate economics** — expected revenue ∝ fraction of total hash power
- **Difficulty adjustment** — every 2016 blocks; stabilizes the 10-minute interval and issuance schedule
- **Reward strategies** — PPS (stable income) vs. PPLNS (long-term participation incentive)
- **Energy cost model** — profit = expected reward − electricity; rising costs drive out small miners
- **Selfish mining strategy** — private chain; timed release to orphan honest blocks
- **Comparison** — selfish mining (reward unfairness) vs. 51% attack (consensus control) vs. double-spending
- **Eyal & Sirer 2014 threshold** — profitability at $\alpha \approx 25\%$; down to ~20% with network advantage

What the team will build

Build a mining simulator with configurable parameters (hash rate distribution, network latency, pool size) that reproduces the selfish mining advantage threshold and visualizes results.

Challenge: Stretch: include a comparison run against a hypothetical 51% attacker so the different threat models are visible side by side.

Wenyi Xie & Rui Yu — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

The Ethereum Virtual Machine (EVM)

Team: Aldo Zhang & Yizhou Liu

EVM architecture: stack-based execution, bytecode format, gas model and metering, execution context (call, delegatecall, staticcall), storage layout, and how smart contracts are deployed and invoked.

What the team presented

- **Why Ethereum needs a VM** — determinism across heterogeneous hardware; state-transition model Y(S,T)=S'
- **Stack machine architecture** — 1024 × 256-bit stack; why 256 matches Keccak-256 and secp256k1
- **Four data locations** — stack/memory/calldata/storage with dramatically different gas costs
- **Gas mechanics** — 21k intrinsic; EIP-2929 cold/warm (2100 vs. 100); SSTORE new-slot 20k
- **Interpreter loop** — PUSH1 / PUSH1 / ADD traced bytecode → stack → gas
- **Contract deployment + call types** — CREATE/CREATE2; CALL vs. DELEGATECALL vs. STATICCALL
- **KEVM formal semantics (Prof. Rosu)** — executable spec that passes the official test suite
- **Roadmap** — EOF (Fusaka 2025); proposed RISC-V transition for ZK provers

What the team will build

Implement a minimal EVM interpreter that correctly executes basic opcodes (arithmetic, comparison, stack manipulation, memory, storage, jumps, and simple calls). Test against known bytecode sequences.

Challenge: Stretch: run your interpreter against KEVM or an official Ethereum test vector and report conformance.

Aldo Zhang & Yizhou Liu — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Smart-Contract Vulnerability Detection with LLMs

Team: Yifan Huang & David Zhao

Compare the EVM with alternative execution environments: WASM-based VMs (Polkadot, Near), MoveVM (Sui, Aptos), and SolanaVM (BPF/SBF). Analyze trade-offs in safety, performance, developer experience, and formal verifiability.

What the team presented

- **Motivation** — $3B+ lost since 2016 — DAO ($60M), Parity ($300M), Euler ($197M)
- **Tool tradeoffs** — static (Slither: fast but shallow) vs. symbolic (Mythril: deep but path-explodes)
- **Euler bug deep-dive** — donateToReserves missing checkLiquidity; static tools can't flag missing code
- **LLM advantages** — semantic understanding (CodeBERT); context awareness (GPTLens); SmartGuard >95%
- **Proposed three-layer architecture** — Layer 1 static → Layer 2 LLM triage → Layer 3 targeted symbolic execution
- **LLM triage walkthrough on Euler** — suspicion score 8/10 + natural-language reasoning flagging the missing check
- **Limitations** — novel attacks, training drift; must complement — not replace — human review

What the team will build

Implement the same simple smart contract (e.g., a token with transfer and balance tracking) on two different VM platforms and benchmark/compare execution cost, deployment size, and developer ergonomics.

Challenge: Stretch: evaluate your detector on a held-out set of real exploits (e.g. Euler, bZx) and report true/false positive rates.

Yifan Huang & David Zhao — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Smart Contract Security and Common Vulnerabilities

Team: Aman Jain & Yu Fu

Taxonomy of smart contract vulnerabilities: reentrancy, integer overflow/underflow, front-running, access control flaws, oracle manipulation, and flash loan attacks. Case studies: The DAO, Parity wallet freeze, Euler Finance.

What the team presented

- **Trust boundaries in 'Code is Law'** — EVM determinism; atomicity enables flash-loan exploits
- **Reentrancy + CEI pattern** — external call before state update enables recursion drain
- **DAO hack 2016** — 3.6M ETH drained; root cause: send before ledger update; Ethereum/ETC split
- **Integer overflow/underflow** — uint256 wrap; BEC batchTransfer 2018 — quadrillions minted
- **Front-running / MEV / sandwich** — mempool visibility + AMM determinism
- **Access control + Parity freeze 2017** — uninitialized library + selfdestruct = 500k ETH bricked
- **Oracle manipulation + Harvest 2020** — Curve pool as implicit oracle; $24M extracted via share-pricing distortion
- **Flash loans + Euler 2023** — $200M exploit via donateToReserves; liquidation-bonus arithmetic

What the team will build

Build a static analysis tool or pattern-based vulnerability scanner that detects common vulnerability patterns (reentrancy, unchecked external calls, tx.origin misuse) in Solidity source code or EVM bytecode.

Challenge: Stretch: ensure your scanner catches all eight vulnerability families covered here in a reproducible test harness.

Aman Jain & Yu Fu — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Smart Contract Security and Common Vulnerabilities

Team: Tse-An Hsu & Yao-Te Wang

Taxonomy of smart contract vulnerabilities: reentrancy, integer overflow/underflow, front-running, access control flaws, oracle manipulation, and flash loan attacks. Case studies: The DAO, Parity wallet freeze, Euler Finance.

What the team presented

- **Reentrancy walkthrough** — code-level Bank + Attacker; drain loop; CEI pattern and reentrancy-lock fixes
- **Integer overflow / underflow** — SafeMath pre-0.8; built-in checks post-0.8; unchecked{} gotchas
- **Front-running** — mempool lifecycle; AMM token pricing; sandwich mechanics
- **Front-running defenses** — slippage tolerance; commit-reveal schemes
- **Access control flaws** — Parity library uninitialized → self-destruct → permanent freeze
- **Oracle manipulation** — staged example: deposit distorts bank → oracle reports $1500 → profit
- **Flash loans + Euler** — borrower + liquidator account split; exploit donateToReserves for $8.8M profit per loop

What the team will build

Build a static analysis tool or pattern-based vulnerability scanner that detects common vulnerability patterns (reentrancy, unchecked external calls, tx.origin misuse) in Solidity source code or EVM bytecode.

Challenge: Stretch: include a negative-result demo — a contract that looks vulnerable but passes, to exercise precision of your scanner.

Tse-An Hsu & Yao-Te Wang — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Automated Market Makers (AMMs)

Team: Julie Lima & Matthew Lynch

AMM design: constant product formula (Uniswap v2), concentrated liquidity (Uniswap v3), impermanent loss, sandwich attacks on AMMs, and comparisons with order-book exchanges.

What the team presented

- **AMM architecture** — LP tokens, traders, smart-contract matching, Uniswap since 2019
- **Constant product formula** — X·Y=K; worked example 10 ETH / 5000 USDC → trade impact
- **Slippage mechanics** — integral along the curve; example shows 11.1% slippage on 1 ETH trade
- **AMM vs. order book** — price discovery, liquidity source, order types, deterministic vs. discrete slippage
- **Impermanent loss** — worked example: ETH $500 → $1000 → LP loses $86 vs. holding
- **Concentrated liquidity (v3)** — LP chooses range; 10–100× capital efficiency; NFT LP positions
- **Sandwich attacks + oracle manipulation** — TWAP as a mitigation; v4 hooks architecture

What the team will build

Implement a simplified AMM smart contract (constant product) with swap, add liquidity, and remove liquidity functions. Deploy on a testnet and demonstrate price impact and impermanent loss scenarios.

Challenge: Stretch: also demonstrate the impermanent-loss curve as prices diverge and overlay buy-and-hold for contrast.

Julie Lima & Matthew Lynch — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

DeFi Lending and Borrowing Protocols

Team: Puan Wang & Ruijie Xu

Lending protocol design: overcollateralization, liquidation mechanics, interest rate models, flash loans, and risk parameters. Analyze architectures of Aave and Compound. Discuss systemic risk and composability.

What the team presented

- **DeFi vs. traditional finance** — peer-to-pool smart contracts replace bank intermediation
- **Overcollateralization** — borrow $100 USDC against $150 ETH; health factor monitors solvency
- **Liquidation mechanism** — external liquidator bots; repay debt + seize collateral at 5–10% discount
- **Risk parameters** — per-asset LTV, liquidation threshold, penalty; blue-chip vs. volatile differentiation
- **Kinked interest-rate model** — APY follows utilization; sharp kink at ~80% enforces liquidity
- **Flash loans** — atomic borrow+use+repay; reverts if unpaid — zero counterparty risk
- **Aave vs. Compound** — aToken rebasing vs. cToken exchange-rate accrual
- **Systemic risks** — oracle reliance; cascading liquidations; composability contagion

What the team will build

Implement a simplified lending protocol smart contract with deposit, borrow, repay, and liquidation functions. Deploy on a testnet and demonstrate a liquidation scenario.

Challenge: Stretch: include a cascading-liquidation stress scenario in your demo and visualize protocol health factor over time.

Puan Wang & Ruijie Xu — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

NFTs and Digital Ownership Standards

Team: Pradnyan Khodke & Pranav Swaminathan

ERC-721 and ERC-1155 standards, metadata architecture (on-chain vs. off-chain, IPFS), marketplace mechanics, royalty enforcement (ERC-2981), and the evolution from PFP collections to utility NFTs (gaming, identity, tickets).

What the team presented

- **NFT identity on Ethereum** — (contract address, token ID); consensus-enforced ownership
- **ERC-721** — core functions ownerOf/transferFrom/tokenURI; approvals for marketplace listings
- **ERC-1155** — multi-type; balance mappings; batch mint/transfer; gaming-inventory use case
- **Metadata storage** — on-chain (permanent, expensive) vs. IPFS (CID-addressed) vs. HTTPS (centralized)
- **Marketplace flow** — setApprovalForAll → on-chain or off-chain signed order → atomic settlement
- **ERC-2981 royalty standard** — spec for querying royalty info; enforcement depends on marketplace
- **Evolution** — PFP collectibles → utility NFTs (tickets, credentials) → soulbound identity tokens

What the team will build

Implement and deploy an ERC-721 contract with on-chain or IPFS-based metadata generation and a simple web frontend for minting and viewing tokens on a testnet.

Challenge: Stretch: include a soul-bound variant in your frontend to show transferability being enforced by contract logic.

Pradnyan Khodke & Pranav Swaminathan — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

NFTs and Digital Ownership Standards

Team: Vipin Gunda & Alex Krysiuk

ERC-721 and ERC-1155 standards, metadata architecture (on-chain vs. off-chain, IPFS), marketplace mechanics, royalty enforcement (ERC-2981), and the evolution from PFP collections to utility NFTs (gaming, identity, tickets).

What the team presented

- **Fungibility taxonomy** — cash (fungible) vs. houses/art (non-fungible); content separated from ownership
- **ERC-721** — unique per-token ID; best fit for art/collectibles/certificates
- **ERC-1155** — multi-type, balance-based; supports fungible + non-fungible + semi-fungible
- **Metadata architecture** — on-chain vs. off-chain; IPFS CID provides content-addressed integrity
- **Marketplace mechanics** — off-chain signed orders (Seaport) save gas; approvals enable transfer
- **Royalty enforcement (ERC-2981)** — standard is informational, not enforcement; marketplace dependency
- **Functional NFT evolution** — identity (ENS), tickets, in-game items, soul-bound credentials

What the team will build

Implement and deploy an ERC-721 contract with on-chain or IPFS-based metadata generation and a simple web frontend for minting and viewing tokens on a testnet.

Challenge: Stretch: include both an ERC-721 and an ERC-1155 asset in your mint flow and show the balance-vs-owner model distinction.

Vipin Gunda & Alex Krysiuk — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Classical BFT Consensus (PBFT and Tendermint)

Team: Anuj Singla & Sahan Yalavarthi

The Byzantine Generals problem, impossibility results (FLP), PBFT protocol (pre-prepare, prepare, commit), and its evolution into Tendermint/CometBFT for blockchain. Discuss message complexity, view changes, and liveness guarantees.

What the team presented

- **Byzantine Generals problem** — agreement + resilience against traitor nodes
- **Network models** — synchronous (DLS, f<n/3) vs. asynchronous vs. partially synchronous
- **FLP impossibility** — safety + liveness both — impossible in fully async with ≥1 crash
- **Partially-synchronous model** — unknown-bound and GST variants as practical middle ground
- **PBFT protocol** — n≥3f+1; pre-prepare / prepare / commit; view-change on primary failure
- **Message complexity** — $O(n^2)$; quadratic scaling is PBFT's main limitation
- **Tendermint improvements** — rotating leader per round; timeout-based rounds; same safety, better liveness
- **Instant finality** — once committed, irreversible — vs. PoW probabilistic finality

What the team will build

Implement a simplified PBFT protocol with configurable node count, simulated message passing, and fault injection (Byzantine and crash faults). Demonstrate safety under f < n/3.

Challenge: Stretch: inject asymmetric Byzantine behavior (e.g. equivocation, silence) and show your protocol correctly performs view-change.

Anuj Singla & Sahan Yalavarthi — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

DAG-Based Consensus

Team: Leah Nicole Ludwikowski & Nithin Parthasarathy

DAG-based approaches to consensus: Narwhal (mempool DAG), Bullshark/Tusk (ordering), and how Sui uses DAG consensus with object-level parallelism. Compare throughput and latency with linear-chain BFT.

What the team presented

- **Consensus refresher** — FLP, CAP, partition-tolerance, Byzantine nodes
- **DAG data structure** — concurrent proposals replace one canonical chain
- **Narwhal — mempool DAG** — round-based with 2f+1 certificates; separates dissemination from ordering
- **Tusk — asynchronous consensus** — random coin derived from the DAG; no extra communication
- **Bullshark — partially-synchronous consensus** — wave-based with two steady-state leaders + fallback
- **Sui as a case study** — object-based state; owned-object fast path skips consensus
- **Throughput comparison** — Narwhal ≈ 160,000 TPS vs. PBFT/HotStuff ≈ 2,000 TPS

What the team will build

Implement a simplified DAG-based ordering protocol with simulated validators that build a DAG of proposals and derive a total order. Demonstrate throughput advantage over sequential consensus.

Challenge: Stretch: vary fault injection (byzantine %, link latency) and show throughput degradation curves vs. a linear BFT baseline.

Leah Nicole Ludwikowski & Nithin Parthasarathy — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Proof of Stake and Validator Economics

Team: Sunwoo Baek & Jeffrey Huang

PoS variants: Casper FFG (Ethereum), Ouroboros (Cardano), and delegated PoS. Cover validator selection, slashing conditions, nothing-at-stake, long-range attacks, and staking economics.

What the team presented

- **PoW → PoS motivation** — energy efficiency; lower barrier to validation
- **Nothing-at-stake + long-range attacks** — validators cheap to equivocate; historical-key rewriting
- **Mitigations** — slashing, key-evolving signatures, weak subjectivity checkpoints
- **Delegated PoS** — representative democracy; Top-N validators; continuous voting
- **Ouroboros (Cardano)** — epochs + slots; VRF-based leader selection; chain-density fork-choice rule
- **Stake pools + delegation** — pooled stake raises selection probability without running a validator
- **Casper FFG** — checkpoint tree; 2/3 supermajority link; justification → finalization
- **Accountable safety** — conflicting finalizations require ≥1/3 slashed stake

What the team will build

Simulate a PoS network with validator staking, block proposal, attestation, and slashing for equivocation. Visualize validator rewards over time under different participation and attack scenarios.

Challenge: Stretch: inject an equivocation attack and visualize which validators get slashed and how accountable-safety is preserved.

Sunwoo Baek & Jeffrey Huang — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Proof of Stake and Validator Economics

Team: Adi Srinivasan & Soham Sarkar

PoS variants: Casper FFG (Ethereum), Ouroboros (Cardano), and delegated PoS. Cover validator selection, slashing conditions, nothing-at-stake, long-range attacks, and staking economics.

What the team presented

- **Consensus requirements** — agreement, validity, termination
- **PoW baseline** — hash-puzzle security; 51% cost; centralization + energy downsides
- **PoS mechanics** — validators stake; selection ∝ stake; slashing for misbehavior
- **Long-chain vs. BFT-style PoS** — gradual finality vs. immediate finality
- **Staking trilemma** — security (high stake), growth (low fees), valuation (low inflation) — pick two
- **Casper FFG (Ethereum)** — 12-sec slots, 32-slot epochs; 67% justify + finalize; coordinated-attack slashing up to 100%
- **Ouroboros + DPoS** — stake-pool RNG; voted delegates with rotation
- **Vulnerabilities + defenses** — nothing-at-stake → slashing; long-range → checkpoints/finality

What the team will build

Simulate a PoS network with validator staking, block proposal, attestation, and slashing for equivocation. Visualize validator rewards over time under different participation and attack scenarios.

Challenge: Stretch: plot the staking-trilemma tradeoffs your simulation makes (e.g. fee vs. inflation vs. slashing fraction).

Adi Srinivasan & Soham Sarkar — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Optimistic Rollups

Team: Akhil Gogineni & Vaasu Kakuturu

Optimistic rollup architecture: off-chain execution, on-chain data availability, fraud proofs, challenge periods, and the sequencer role. Cover Optimism and Arbitrum designs and their trust assumptions.

What the team presented

- **Ethereum L1 scaling gap** — 15–30 TPS; need higher throughput with L1-inherited security
- **Optimistic rollup core idea** — execute off-chain; post data + state root; 7-day challenge window
- **Architecture** — sequencer batches and commits; challengers re-execute and submit fraud proofs
- **Fraud-proof protocols** — non-interactive vs. interactive (binary search down to a single instruction)
- **Arbitrum BoLD vs. Optimism Cannon** — EVM-native vs. custom MIPS-based dispute VM
- **Economic incentives** — bonds on both sides; wrong party is slashed — why fraud is rare in practice
- **Data availability** — EIP-4844 blobs drastically reduce DA cost vs. calldata
- **Optimistic vs. ZK rollups** — crypto-economic vs. mathematical; scalability vs. immediate finality

What the team will build

Implement a simplified optimistic rollup: a smart contract that accepts batched state commitments and processes fraud proof challenges that can revert invalid state transitions.

Challenge: Stretch: run the interactive fraud proof down to the single disputed step and show the dispute resolution on-chain.

Akhil Gogineni & Vaasu Kakuturu — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Bitcoin Mining and Selfish Mining

Team: Zihao Chen & Huyang Yu

Mining economics: hash rate, difficulty adjustment, pool strategies (PPLNS, PPS), energy consumption, and the selfish mining attack (Eyal & Sirer, 2014). Analyze the threshold at which selfish mining becomes profitable.

(Team presented Bitcoin mining instead of their assigned zk-Rollups topic.)

#topic-13b-zk-rollups

What the team presented

- **Nakamoto consensus** — longest-chain rule; propagation delay naturally creates forks + stale blocks
- **Mining economics** — hash rate as guesses/sec; difficulty retargeting every 2016 blocks
- **Pool strategies** — PPS (pool absorbs risk) vs. PPLNS (payouts only on found blocks)
- **Selfish-mining exploit** — withhold blocks; release strategically to force honest blocks to become stale
- **State machine** — decision based on lead gap Δ: reset, race, knockout, cat-and-mouse
- **Threshold math** — $\alpha^* \geq 1/3$ at $\gamma=0$, dropping toward ~25% as $\gamma \to 1$ — below the 'safe' 51% line

What the team will build

Build a mining simulator with configurable parameters (hash rate distribution, network latency, pool size) that reproduces the selfish mining advantage threshold and visualizes results.

Challenge: Stretch: sweep γ from 0→1 and α from 0→50% and visualize where selfish mining becomes profitable as a 2-D heatmap.

Zihao Chen & Huyang Yu — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Blockchain Bridges and Cross-Chain Communication

Team: Brandon Che Li & Yahav Mangel

Bridge architectures: lock-and-mint, message passing, light client verification, and optimistic bridges. Trust models and notable exploits (Ronin, Wormhole, Nomad). Emerging standards (IBC, LayerZero).

What the team presented

- **Why bridges exist** — isolated chains with own consensus; users expect seamless asset transfer
- **Implementation challenges** — state isolation, scalability, extra attack surface
- **HTLC naive swap** — hash + timelock; simple + fast but griefing-vulnerable and un-scalable
- **Lock-and-mint pegged assets** — wrapped ERC-20 on destination; requires trusted validator in between
- **Arbitrary message bridges** — generalized remote function execution; wider attack surface
- **Trust models** — external validators vs. light clients vs. optimistic verification — cost/security tradeoff
- **Security failures** — Qbridge $80M — deposit logic failed to verify source-chain lock
- **LayerZero modular design** — endpoints; DVNs for verification; execution split out for flexibility

What the team will build

Implement a simplified bridge between two local test chains that locks tokens on one chain and mints wrapped tokens on the other, using a relayer to pass attestations.

Challenge: Stretch: demonstrate one of the trust-model failure modes (e.g. unlock-without-lock) and show your design preventing it.

Brandon Che Li & Yahav Mangel — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Blockchain Bridges and Cross-Chain Communication

Team: Liuyuan Zhi & Baige He

Bridge architectures: lock-and-mint, message passing, light client verification, and optimistic bridges. Trust models and notable exploits (Ronin, Wormhole, Nomad). Emerging standards (IBC, LayerZero).

What the team presented

- **Bridge ecosystem scale** — $25B+ bridged cross-chain; $2B+ lost to misdesign
- **Three core mechanics** — lock-and-mint, message passing, light-client verification
- **Validator-based designs** — Wormhole 19 guardians (VAA); LayerZero oracle + relayer split; Axelar PoS
- **Light-client bridges** — IBC as gold standard; on-chain consensus-state verification; Merkle/BFT proofs
- **Optimistic bridges** — assume valid, 7-day challenge window; Connext+Across bridging UX
- **Major hacks walkthrough** — Ronin $625M (5-of-9 validator compromise); Wormhole $320M (sig forge); Nomad $190M (init bug)
- **Common attack patterns** — multisig compromise, logic bugs, flash-loan manipulation
- **Takeaways** — trustlessness (IBC) vs. reach (LayerZero); ZK bridges as the next frontier

What the team will build

Implement a simplified bridge between two local test chains that locks tokens on one chain and mints wrapped tokens on the other, using a relayer to pass attestations.

Challenge: Stretch: include a before/after security test showing your bridge correctly rejects one of the above attack patterns.

Liuyuan Zhi & Baige He — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Maximal Extractable Value (MEV)

Team: Swara Kurakula & Carlos Andres Delgado Urbanez

MEV taxonomy: front-running, sandwich attacks, arbitrage, liquidations. Impact on users and network health. Mitigation: Flashbots, MEV-Share, proposer-builder separation (PBS), encrypted mempools.

What the team presented

- **MEV definition** — value captured by controlling ordering, inclusion, exclusion within a block
- **Centralized vs. decentralized** — internalized in CEX; exposed + extractable on-chain
- **Actors** — users, searchers, builders, proposers — the modern MEV supply chain
- **Attack types** — front-running, sandwich attacks, arbitrage, liquidations
- **Negative consequences** — hidden user tax; centralization pressure; chain-reorg incentive
- **Flashbots mitigations** — MEV-Boost / MEV-Share; sealed auctions; proposer-builder separation
- **PBS architecture** — searchers → builders → relays → proposer; signed header commits blindly
- **Encrypted mempools** — threshold encryption via keypers; decrypt only after ordering

Topic 15a · Maximal Extractable Value (MEV)

What the team will build

Build an MEV detection tool that analyzes a set of transactions (historical or simulated) to identify sandwich attacks, arbitrage patterns, or front-running. Visualize extracted value.

Challenge: Stretch: classify historical mainnet transactions and quantify the MEV extracted per day over a target window.

Swara Kurakula & Carlos Andres Delgado Urbanez — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Maximal Extractable Value (MEV)

Team: Armaan Patel & Varad Rasalkar

MEV taxonomy: front-running, sandwich attacks, arbitrage, liquidations. Impact on users and network health. Mitigation: Flashbots, MEV-Share, proposer-builder separation (PBS), encrypted mempools.

What the team presented

- **PoS + mempool context** — validators selected by stake; mempool as visible waiting room
- **MEV as an invisible tax** — ordering decisions turn into price-impact profits
- **Attack taxonomy** — front-running, back-running, sandwich attack (combined)
- **'Good MEV'** — arbitrage keeps prices consistent; liquidation keeps lending protocols solvent
- **User + network harms** — higher gas, worse execution, small-trader disadvantage
- **Flashbots + MEV-Boost** — private channel; sealed auction; front-running protection
- **MEV-Share** — programmable privacy + 90% rebate to users via back-running bundles
- **PBS + encrypted mempools** — decentralize role of proposer/builder; hide content entirely

What the team will build

Build an MEV detection tool that analyzes a set of transactions (historical or simulated) to identify sandwich attacks, arbitrage patterns, or front-running. Visualize extracted value.

Challenge: Stretch: include a comparison on the same workload of sandwich profits with and without an encrypted mempool.

Armaan Patel & Varad Rasalkar — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Weak Consensus and Scalable Settlement Protocols

Team: Vikramaditya Pidaparthi & Chen Li

Protocols that achieve secure, decentralized payment/settlement finality without full blockchain consensus: FastPay, Linera (micro-chains), POD, and FastSet (Chen & Rosu, 2025). Explain why strong consistency and total ordering are unnecessary for many financial operations, how these protocols capitalize on massive parallelism with no inter-validator communication, and how they compare to blockchains on security, decentralization, and scalability. FastSet, the most recent and comprehensive, generalizes beyond payments to claims including asset transfers, shared data updates, and more, with formal correctness proofs.

What the team presented

- **FastPay** — account sequence numbers; BCB committee vote → certificate → settlement
- **Linera microchains** — single-owner / permissioned / public chains; one validator set processes all
- **Pod** — data structure + protocol without inter-replica communication; weaker ordering
- **FastSet (Prof. Rosu + PiSquared)** — generalizes claims beyond payments; formally proven correctness
- **Weak independence** — operations commute except within the same actor — no global order needed
- **Seven-step FastSet protocol** — submit → validate → pending → quorum cert → pre-settle → settle
- **Correctness guarantees** — security, determinism, monotonicity, liveness — all formally proven

What the team will build

Implement a simplified FastPay- or FastSet-style protocol where a set of validators independently verify and certify owner-signed claims (e.g., transfers). Demonstrate instant finality and horizontal scalability without a global ledger or inter-validator communication.

Challenge: Stretch: benchmark scaling by sweeping 4→8→16→32 validators and show throughput scales linearly.

Vikramaditya Pidaparthi & Chen Li — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.

Weak Consensus and Scalable Settlement Protocols

Team: Kushal Agarwal & Ameya Kolhatkar

Protocols that achieve secure, decentralized payment/settlement finality without full blockchain consensus: FastPay, Linera (micro-chains), POD, and FastSet (Chen & Rosu, 2025). Explain why strong consistency and total ordering are unnecessary for many financial operations, how these protocols capitalize on massive parallelism with no inter-validator communication, and how they compare to blockchains on security, decentralization, and scalability. FastSet, the most recent and comprehensive, generalizes beyond payments to claims including asset transfers, shared data updates, and more, with formal correctness proofs.

What the team presented

- **Motivation: AI-agent payments** — x402 HTTP 402 for agent-to-API payments — needs internet-speed settlement
- **Throughput gap** — Bitcoin 7 TPS, Ethereum 30 TPS, Visa 65k TPS — blockchains don't clear the bar
- **Full consensus vs. payment needs** — $O(n^2)$ inter-validator communication is overkill for transfers
- **Spectrum placement** — weak consensus sits between trustless blockchain and centralized payment providers
- **FastPay, Linera, Pod** — comparative lineage and design tradeoffs
- **FastSet (Prof. Rosu + PiSquared)** — extends to arbitrary claims; 100k+ TPS; sub-100ms global finality
- **Common insight** — eliminate inter-validator communication, drop from $O(n^2)$ to $O(1)$
- **Intended planned project** — three-layer simulator: x402 ↔ facilitator bridge ↔ FastSet validator network

What the team will build

Implement a simplified FastPay- or FastSet-style protocol where a set of validators independently verify and certify owner-signed claims (e.g., transfers). Demonstrate instant finality and horizontal scalability without a global ledger or inter-validator communication.

Challenge: Stretch: confirm Byzantine tolerance up to f faults in your simulator and plot finality latency vs. payload size.

Kushal Agarwal & Ameya Kolhatkar — please add your implementation specifics: language/stack, architecture choices, concrete metrics you plan to measure.