CS 521

Foundations of Crypto: From Blockchain to Present

Course Structure and Syllabus — Spring 2026

Grigore Rosu

The Grainger College of Engineering • University of Illinois

Course Overview

CS 521 explores the foundations of crypto, from the history of money and cryptographic primitives through Bitcoin, Ethereum, consensus mechanisms, and modern developments beyond traditional blockchains. The course combines **instructor-led lectures** on core topics with **student-driven pair projects** that deepen understanding through research, presentation, and implementation.

Logistics

Schedule

Day	In Person	Online
Wednesdays	Urbana (~45 students)	~15 students
Fridays	Chicago (~15 students)	~45 students

The course runs in hybrid format across both campuses. All lectures and presentations are accessible to both in-person and online students.

Course Structure

The course has two main components: **instructor lectures** covering foundational material, and a **semester-long pair project** combining research, presentation, and coding on a single topic.

Component 1: Instructor Lectures

The instructor will deliver lectures on core topics throughout the semester, including:

- Short History of Money
- Basic Crypto Primitives
- Bitcoin
- Ethereum and Smart Contracts
- Consensus Mechanisms
- Scaling and Layer 2 Solutions
- DeFi, NFTs, and Applications
- Security, Attacks, and Formal Verification
- Beyond Blockchains: Weak Consensus and Scalable Settlement

These lectures provide the shared foundation that all students are expected to master.

Component 2: Pair Project

Students form **pairs (groups of 2)**. In exceptional cases, a group of 3 may be approved, but pairs are strongly preferred. Each pair selects one topic from the curated list (see Appendix A). The same pair works on **both the presentation and the coding project** for that topic throughout the entire semester.

Each topic comes with a defined scope for both the conceptual presentation and the coding implementation, ensuring both dimensions have sufficient depth. Some topics offer a/b variants (e.g., 2a: zk-SNARKs vs. 2b: zk-STARKs), giving pairs on the same topic number a choice of focus.

Topic sharing: Since there are approximately 30 pairs and 23 topic options, some topics will have two pairs. Each pair works independently, but awareness of the other pair's approach is encouraged. Presentations and implementations must be distinct.

Semester Timeline

Period	Activity	Deliverables
Weeks 1–4	Foundations	Instructor lectures on core topics. Pairs form and select topics from the curated list. All deadlines: Sunday midnight AoE.
Week 6	First Presentation	Each pair submits a recorded presentation (10–12 min) providing a conceptual overview of their topic. Captions required. Post 3 questions in your channel.
Week 7	Reflection #1	Individual written reflection (2 pages): connect your topic to the foundational material from lectures. All students answer cross-topic questions from Round 1.
Weeks 5–11	Lectures + Implementation	Instructor continues lectures. Pairs build their coding project. Brief progress check-in with instructor.
Week 12	Second Presentation	Improved recorded presentation (10–12 min) incorporating feedback, plus a live demo of the coding project. Captions required. Post 3 new questions.
Week 13	Reflection #2	Individual written reflection (2 pages): technical challenges, design decisions, and lessons learned. All students answer cross-topic questions from Round 2.
Weeks 14–15	Live Showcase	Selected pairs (6–8) present live to the full class. Serves as a showcase of the best and most interesting projects.
Week 16	Final Deliverables	Final report (pair) + Reflection #3 (individual, 2

		pages): critical self-evaluation. Pairs post 3 final questions. All students answer Round 3.

Deliverables in Detail

Recorded Presentations

- Each pair submits two recorded presentations over the semester: an initial conceptual overview (Week 6) and an improved version with a coding demo (Week 12).
- Presentations should be 10–12 minutes in length and must include captions (auto-generated is fine).
- All presentations will be made available to the entire class as a shared learning resource.
- The instructor will review all presentations and provide written feedback. Selected presentations may be invited for a live showcase in Weeks 14–15.

Coding Project

- The coding project implements concepts from your chosen topic. The scope is defined in the curated topic list (Appendix A).
- Code must be maintained in a Git repository with meaningful commit history from both partners. The commit history serves as evidence of individual contribution.
- A live demo is included in the second recorded presentation (Week 12).

Individual Reflections

Each student submits **three individual written reflections** over the semester (2 pages each). These are the primary mechanism for individual assessment. Reflections must demonstrate your personal understanding and engagement with the material.

- Reflection #1 (Week 7): Connect your project topic to the foundational course material. Demonstrate understanding of how the crypto primitives, blockchain structures, and concepts from lectures relate to your specific topic.
- Reflection #2 (Week 13): Discuss the technical challenges encountered during implementation, design decisions made, and what you learned from building the code.
- Reflection #3 (Week 16): Critically evaluate your own contribution to the pair project. What would you do differently? What did you learn about the topic that surprised you?

Note on AI usage: You may use AI tools to assist with writing. However, reflections that demonstrate only surface-level understanding will receive lower grades regardless of writing quality. The purpose is to show *your* understanding, not to produce polished prose.

Final Report

- Each pair submits one final report (approximately 10 pages) combining a conceptual synthesis of the topic with a technical description of the implementation.
- The report should incorporate all feedback received over the semester and serve as a comprehensive reference on the chosen topic.
- Both partners are credited equally on the report; individual differentiation comes from the reflections and git history.

Grading

Component	Weight	Type
Recorded Presentations (2)	**25%**	Pair
Coding Project + Git History	**25%**	Pair*
Individual Reflections (3)	**30%**	Individual
Final Report	**10%**	Pair
Cross-Topic Participation	**10%**	Individual

***Coding Project:** While the project is a pair deliverable, the git commit history will be reviewed to assess individual contribution. Significant imbalance in contributions may result in different grades for each partner.

Cross-Topic Participation

A key goal of this course is for every student to develop broad understanding across *all* topics, not just their own. To encourage this, the course uses a structured **cross-topic Q&A system** run through the class Slack workspace.

Slack Workspace Structure

- Each pair has a dedicated public Slack channel for their topic (e.g., #topic-05a-lightning, #topic-16-weak-consensus).
- Pairs post all their deliverables (recorded presentations, reports, demos) in their channel.
- All channels are visible to all students. You are expected to follow and engage with channels beyond your own.

How It Works

Each time a pair posts new material in their channel (first presentation, second presentation, final report), they also post **3 questions** about their material. These questions should require genuine engagement with the posted content — they cannot be answered without watching the video or reading the report.

Other students answer these questions by submitting responses via a Google Form linked from the channel. Submissions are pseudonymous (see below). The pair then grades the responses they receive.

Pseudonymous Submissions

To ensure blind grading, all cross-topic answers are submitted under a **pseudonym** assigned to you by the instructor. Your pseudonym is a **64-character hexadecimal string** (derived from a cryptographic hash). It stays the same for the entire semester.

The instructor will DM each student their pseudonym on Slack. The mapping between pseudonyms and real identities is kept confidential by the instructor. Grading pairs see only your pseudonym, never your identity.

- Use your assigned pseudonym in all cross-topic answer submissions.
- Do not share your pseudonym with other students.
- If you lose your pseudonym, ask the instructor to re-send it via Slack DM.

Answering Questions

- Every student must answer the questions posted by every other topic channel in each round (3 rounds across the semester).
- Answers are submitted via the Google Form linked in each channel, using your pseudonym.
- Answers should demonstrate genuine understanding of the material, not surface-level responses.
- Expect to spend roughly 1–1.5 hours per week on cross-topic engagement (watching presentations, reading reports, and answering questions). This is a significant and

intentional part of the course — by the end of the semester, you will have meaningful exposure to every topic, not just your own.

Grading Answers

- Each pair grades the pseudonymous answers they receive for their own questions.
- Grading uses a simple 0/1/2 scale: 0 = did not engage or incorrect, 1 = partial understanding, 2 = solid understanding.
- Pairs submit their grades to the instructor via a spreadsheet. The instructor maps pseudonyms to students at the end of the semester.
- Grading is expected to be fair and consistent. The instructor may spot-check gradings.

Participation Score

Your cross-topic participation score (10% of final grade) is computed from the total points you accumulate across all question sets in all three rounds. Students who consistently demonstrate thoughtful engagement with other teams' material will earn the highest scores.

Policies

Pair Formation

- Pairs should be formed by Sunday midnight AoE, end of Week 4. Students may form pairs across campuses (Chicago/Urbana), but should be aware this means fully remote collaboration.
- If you cannot find a partner, contact the instructor for assistance.
- In exceptional circumstances, a group of 3 may be approved. Solo projects are not permitted.

Topic Selection

- Topics are selected from the curated list (Appendix A) on a first-come, first-served basis.
- Each topic option (including a/b variants) can be taken by at most two pairs. Pairs on the same topic work independently.
- Topic selection opens in Week 4 on a first-come, first-served basis. Deadline: Sunday midnight AoE, end of Week 4. Pairs who have not selected a topic by the deadline will be assigned one automatically.

Pair Issues

If a pair experiences collaboration difficulties, contact the instructor before Week 5. After the first presentation, restructuring pairs becomes significantly harder. Early communication is essential.

Academic Integrity

All individual reflections must represent your own understanding. Code must be original work by the pair, with appropriate attribution for any external libraries or references used. AI tools may be used as assistants for both writing and coding, but you are responsible for understanding and being able to explain everything you submit.

Late Submissions

All deadlines are Sunday midnight AoE (Anywhere on Earth) at the end of the specified week. Late submissions receive a 10% penalty per day, up to 3 days. After 3 days, the submission receives zero credit. Extensions may be granted for documented emergencies.

Appendix A: Curated Topic List

The following 16 topics (23 options including a/b variants) are available for pair projects. Each entry defines the expected scope for both the **presentation** and the **coding project**. Topics marked with a/b offer two focus areas; pairs selecting the same topic number choose different variants.

Topic 1: Merkle Patricia Tries

Presentation: The Merkle Patricia Trie data structure, its role in Ethereum state management (accounts, storage, transactions, receipts), and comparison with simple Merkle trees and other authenticated data structures.

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

Topic 2a: zk-SNARKs

Presentation: Zero-knowledge proof fundamentals with focus on zk-SNARKs: trusted setup, bilinear pairings, Groth16, and applications in privacy-preserving blockchains (Zcash). Discuss trust assumptions and the ceremony process.

Coding project: Use a library such as circom/snarkjs to design, compile, and verify a ZKP circuit (e.g., proving knowledge of a hash preimage or a valid Sudoku solution without revealing it).

Topic 2b: zk-STARKs

Presentation: 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.

Coding project: 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.

Topic 3: Hash-Based Accumulators and Authenticated Data Structures

Presentation: RSA accumulators, Merkle accumulators, and vector commitments. Their use in stateless blockchain clients, certificate transparency, and revocation. Compare efficiency and trust assumptions.

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

Topic 4: Bitcoin Script and Transaction Types

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

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

Topic 5a: Bitcoin Payment Channels and the Lightning Network

Presentation: 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.

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

Topic 5b: Generalized State Channels

Presentation: Generalized state channels beyond payments: off-chain execution of arbitrary state transitions, dispute resolution, virtual channels, and state channel networks. Cover frameworks such as Perun, Nitro, and concepts behind Kite AI.

Coding project: Implement a generalized state channel framework where two parties execute a simple application (e.g., tic-tac-toe or a token swap) off-chain, with on-chain dispute resolution via a smart contract deployed on a testnet.

Topic 6: Bitcoin Mining and Selfish Mining

Presentation: 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.

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

Topic 7a: The Ethereum Virtual Machine (EVM)

Presentation: 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.

Coding project: 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.

Topic 7b: Alternative Blockchain Virtual Machines

Presentation: 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.

Coding project: 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.

Topic 8: Smart Contract Security and Common Vulnerabilities

Presentation: 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.

Coding project: 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.

Topic 9a: Automated Market Makers (AMMs)

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

Coding project: 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.

Topic 9b: DeFi Lending and Borrowing Protocols

Presentation: 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.

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

Topic 10: NFTs and Digital Ownership Standards

Presentation: 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).

Coding project: 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.

Topic 11a: Classical BFT Consensus (PBFT and Tendermint)

Presentation: 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.

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

Topic 11b: DAG-Based Consensus

Presentation: 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.

Coding project: 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.

Topic 12: Proof of Stake and Validator Economics

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

Coding project: 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.

Topic 13a: Optimistic Rollups

Presentation: 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.

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

Topic 13b: ZK-Rollups

Presentation: ZK-rollup architecture: validity proofs, proof generation pipelines, on-chain verification, and data availability. Cover zkSync and StarkNet designs. Compare finality and cost with optimistic rollups.

Coding project: Implement a simplified zk-rollup verifier: a contract that accepts batched transactions with a validity proof (using a simple ZKP scheme) and updates state only if the proof verifies.

Topic 14: Blockchain Bridges and Cross-Chain Communication

Presentation: 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).

Coding project: 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.

Topic 15: Maximal Extractable Value (MEV)

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

Coding project: 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.

Topic 16: Weak Consensus and Infinitely Scalable Settlement Protocols

Presentation: 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.

Coding project: 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.