CS251 Final Exam, Winter 2025

Tuesday, Dec. 9, 2025

Instructions:

- You must do the exam on your own. You may not collaborate with others. Students are bound by the Stanford honor code.
- The exam is open book, but not open AI. You may not use a search engine or an AI bot when answering the questions.
- To submit your answers either (i) use the provided LaTeX template (see the link in the next bullet), or (ii) write your answers on the PDF of the exam, or (iii) write your answers on blank sheets of paper, but please make sure to start each question on a new page. When done, please upload your solutions to Gradescope (**KDJ7NE**). You have an extra 15 minutes to do so. There is no need to upload this first page of the exam.
- The LaTeX template is at https://cs251.stanford.edu/final-71pbntq39.zip. Please do not share the link with others.

- The exam has 6 questions totaling 100 points.
- You have three hours to complete them.
- Please keep your answers concise.

1	/24
2	/12
3	/15
4	/20
5	/14
6	/15
Total	/100

Problem 1. [24 points]: Questions from all over.

A) The recent Fusaka Ethereum upgrade raised the maximum gas per block from 45M to 60M. What is the benefit of raising the max gas per block?

B) Why stop at 60M? What would go wrong if Ethereum set the max gas per block to a trillion?

C) In Lecture 19 we looked at Bridging protocols. Consider a lock-and-mint bridge that is used to move assets from a chain N without finality (e.g., using Nakamoto-style consensus) to a chain P with finality (e.g., using a PBFT-style consensus). A user on chain N sends their assets to the bridge address on N, and asks the bridge contract on P to mint tokens equivalent to the amount locked up on N. The bridge contract on P sends the newly minted tokens to the user on chain P, and that transaction is finalized on P. What would go wrong if at that point chain N had a re-org and the user's transaction that sent the assets to the bridge address on N is no longer part of the longest chain? How can chain P protect itself from this?

D) A mapping in Solidity stores key-value pairs, much like a dictionary in Python. In Python the `keys()` methods returns a list of all the keys in the dictionary. Please explain why Solidity does not provide a `keys` function for a mapping, as in Python.
Hint: Think how a mapping in Solidity is implemented in the EVM.

E) A Rollup system can provide a much higher transaction throughput than Ethereum. What is the key reason that a Rollup can process more transactions per second than Ethereum?

F) In the very last lecture we talked about DePIN. What is DePIN and how does it relate to blockchains?

Problem 2. [12 points]: Proof-of-Stake Consensus.

In Lecture 6 we looked at a simple PBFT-style proof-of-stake consensus protocol. Recall that in every slot a leader is chosen, the leader broadcasts a block to the network, and once two-thirds of the validators sign the block, the block is finalized at that slot.

A) Suppose that N validators are currently staked and sign blocks on every slot. However, after some global disaster (an alien invasion?) 90% of the validators can no longer sign blocks, ever, while the remaining 10% of the validators continue to function as normal. What happens to the Ethereum network? Recall that missing many slots triggers a slashing event, and once a validator loses all of its stake it is kicked out of the validator set. How many defunct validators need to be kicked out before the network can again finalize blocks? You can assume that all validators staked the same minimum amount, namely 32 ETH.

How to choose a leader at each slot? Recall that validator j locks up s_j ETH, for some integer $s_j \geq 32$. Let S be the sum of s_j over all validators — the total locked stake. Validator j should be elected as a leader once every S/s_j slots, in expectation. The intent is that the more stake a validator locks up, the more frequently they will be chosen as the leader. For simplicity, let's assume that S/s_j is an integer for all j.

B) Let's consider two ways to elect a leader. In the first method, at the beginning of every slot the chain chooses a verifiable uniform random value $r_j \in \{1, \ldots, S/s_j\}$ for every validator (that is, one random value is chosen per validator). Then validator j acts as a leader for that slot if $r_j = 1$. Explain if this is a good way to elect a leader.

C) In the second method, a single verifiable uniform random value $r \in \{0, \ldots, S-1\}$ is chosen at the beginning of every slot, and validator j acts as the leader if r is in the interval $A \leq r < A + s_j$, where $A = \sum_{k<j} s_k$. Explain if this is a good way to elect a leader.

Problem 3. [15 points]: DeFi.

A) In Lecture 10 we explored an AMM governed by the constant product formula. Consider a pool that holds X tokens of type A and Y tokens of type B. Alice has a tokens of type A and wants to exchange them for tokens of type B. She is considering two strategies:

- Send all her a tokens of type A in one transaction and get back b tokens of type B (we showed that the AMM sends back $b = a \cdot Y/(X + a)$ tokens), or
- Send $a/2$ tokens to get back b_1 tokens in one transaction. Later send the remaining $a/2$ tokens and get back another b_2 tokens in another transaction. Overall Alice gets back $b_1 + b_2$ tokens of type B.

Which method is more advantageous for Alice? That is, is $b > b_1 + b_2$ or $b < b_1 + b_2$ or $b = b_1 + b_2$? You may assume that Alice is the only one interacting with the pool and that there are no additional fees (i.e. $\phi = 1$). Please justify your answer.
Hint: There is a two line answer to this question.

B) Recall that an exchange that is using a CLOB (central limit order book) takes buy and sell requests from customers, matches up the buy and sell orders, and executes the resulting list of transactions on chain. On Ethereum, a fully on-chain CLOB (i.e., where the matching algorithm runs in a smart contract) is not common. Why is that? Note that on other chains (such as Hyperliquid) CLOBs are widely used.

C) Suppose that an ERC-20 smart contract for the XYZ token includes a `pause` function that pauses all activity in the contract until a `resume` function in called. Once `pause` is called and successfully pauses all activity in the ERC-20 contract, does that prevent the public from buying and selling this XYZ token, for example on an exchange like Coinbase? If so explain why, if not explain why not.

Problem 4. [20 points]: Solidity programming.

The state of California decides to implement its lottery system as an Ethereum contract. The contract should support the following methods:

- buyTicket: any user can send the price of a ticket in Ether to the contract and the contract will record that user's address.
- doLottery: this method is called by the state lottery system once a week to randomly select that week's winner, if any. If there is a winner, the contract sends 90% of the pot to the winner's address and the remaining 10% rolls over to the following week. If there is no winner, the entire pot rolls over to the following week. Either way, the set of users resets to the empty set.

The contract proceeds in epochs, where each epoch is seven days, starting from the moment that the lottery contract is created. Say n users participate in a particular epoch. Each user is assigned an ID between 0 and $n-1$ in sequential order.

The doLottery method can only be called by the state of California within a 10 minute window after the end of each epoch. The method selects a winner by computing the current block hash modulo $2n$, and if the number matches a user ID, that user is the winner. Otherwise there is no winner, which happens with probability 1/2. The block hash modulo $2n$ can be computed as `(blockhash(block.number)%(2*n))`.

A) Write the solidity code to implement this contract. Either use a blank sheet or the provided LaTeX template. Recall that in Solidity, the global variable `block.timestamp` refers to the current time (specified in seconds), and `block.timestamp + 7 days` and `block.timestamp + 10 minutes` refer to the current time plus the specified offset.

B) Is this a good idea? Are there parties that can manipulate the lottery to greatly increase their chances of winning? If so explain how, if not explain why not.

Problem 5. [14 points]: Proof systems.

Succinct proof systems (SNARKs) play a crucial role in on-chain privacy as well as in scaling solutions and bridging. Recall that in Lecture 16 we used a *polynomial commitment scheme* (PCS) to build a SNARK.

- First, the prover commits to a tableau that represents the computation trace of a circuit. It does so by interpolating a univariate polynomial $f \in \mathbb{F}_p[X]$ of degree at most d so that $f(\omega_0), \ldots, f(\omega_d)$ are the values in the tableau. Here p is some large prime and $\omega_0, \ldots, \omega_d$ are some fixed elements in $\mathbb{F}_p$. The prover sends to the verifier a commitment to f.
- Second, the prover proves that the tableau encoded by the committed polynomial represents a valid computation trace.

One of the main tools used in the second step is a **ZeroCheck proof system** that lets the prover convince the verifier that a committed polynomial $g \in \mathbb{F}_p[X]$, possibly of degree much greater than d, is identically zero on the set $\Omega := \{\omega_0, \ldots, \omega_d\}$. That is, $g(\omega_0) = g(\omega_1) = \cdots = g(\omega_d) = 0$. As short hand we write $g(\Omega) = 0$. In this problem we will use ZeroCheck as a black box to build proof systems for other properties of a committed polynomial.

A) Let $h \in \mathbb{F}_p[X]$ be a committed polynomial. In this part our goal is to build a proof system that lets the prover convince the verifier that all the evaluations $h(\omega_0), \ldots, h(\omega_d)$ are in $\{0, 1\}$. As short hand we write $h(\Omega) \subseteq \{0, 1\}$. This question comes up often when proving real-world computations. Show how the prover can prove that $h(\Omega) \subseteq \{0, 1\}$ using a single invocation of ZeroCheck.
Hint: Observe that a polynomial commitment to h is also a polynomial commitment to the polynomial $w := h^2$. In particular, for some $u, v \in \mathbb{F}_p$, the prover can convince the verifier that $w(u) = v^2$ by providing an evaluation proof that $h(u) = v$.

B) Generalize your method from part (A) to show how the prover can convince the verifier that $h(\Omega) \subseteq \{0, 1, 2, \ldots, 7\}$ using a single invocation of ZeroCheck.
Discussion: There are more efficient proof systems for part (b), but we will leave that for another day.

C) Finally, suppose we are given a PCS that can commit to polynomials in $\mathbb{F}_p[X]$ of degree at most d. Our goal is to build a PCS that can commit to polynomials of higher degree, in particular of degree at most $2d+1$. To do so, observe that if g has degree at most $2d+1$, then it can be decomposed into two polynomials g_{high} and g_{low}, each of degree at most d, such that $g(X) = X^{d+1} g_{high}(X) + g_{low}(X)$. For example, with $d = 2$,

$$\text{for} \quad g(X) = 5X^5 + 4X^4 + 3X^3 + 2X^2 + X - 7$$

$$\text{we have} \quad g_{high}(X) = 5X^2 + 4X + 3 \quad \text{and} \quad g_{low}(X) = 2X^2 + X - 7,$$

$$\text{and indeed} \quad g(X) = X^3 g_{high}(X) + g_{low}(X).$$

To commit to g the committer uses the provided PCS to commit to g_{high} and g_{low}. That is, it commits to g by sending two commitments $(\mathsf{com}_h, \mathsf{com}_l)$, one for g_{high} and one for g_{low}. Now, let $u, v \in \mathbb{F}_p$. Explain how the prover can convince the verifier that $g(u) = v$.

Hint: Observe that for a polynomial g, and the derived polynomials g_{high}, g_{low}, we have that $g(u) = v$ if and only if there are $v_h, v_l \in \mathbb{F}_p$ such that

$$g_{high}(u) = v_h, \quad g_{low}(u) = v_l, \quad \textit{and} \quad v = u^{d+1} v_h + v_l$$

- The proof π that $g(u) = v$ is a tuple of four elements. How does an honest prover construct this 4-tuple given (g, u, v) as input?

- The Verifier takes $\big(\mathsf{com}_g = (\mathsf{com}_h, \mathsf{com}_l), u, v, \pi\big)$ as input. What does it do to check the proof π claiming that $g(u) = v$?

Problem 6. [15 points]: Rollups.

In Lecture 18 we discussed a way to scale the Ethereum transaction rate using a layer 2 Rollup (or Rollup for short). We saw two types of Rollups: an optimistic Rollup and a zk-Rollup. In this question we consider a centralized Rollup coordinator that accepts transactions from the public, sequences the submitted transactions into Rollup blocks, executes one block after another to update the Rollup state, and submits state updates to the underlying chain.

A Rollup usually pushes its state updates to one underlying chain, such as Ethereum. But suppose a zk-Rollup maintains smart contracts on two chains, say Ethereum and Solana. The coordinator periodically pushes a state update (i.e., a Merkle root) and proofs to both layer 1 chains. Doing so lets users who have assets on both Ethereum and Solana easily move their assets to the Rollup. This lets Alice buy something using her combined ETH and SOL balance on the Rollup. Without this type of Rollup, combining the balances would require some form of bridging. Note that there are two Rollup smart contracts, one on each chain. The Ethereum contract holds the Rollup assets on Ethereum and the Solana contract holds the Rollup assets on Solana. These contracts hold the assets on behalf of the Rollup users.

A) In the honest case, the coordinator always pushes the same Merkle root to both chains. Can a malicious coordinator push one valid Merkle root and proof to one chain, and a different valid Merkle root and proof to the other chain? In a zk-Rollup, how would a malicious coordinator create two Merkle roots along with their valid update proofs?
Hint: Recall that the coordinator chooses the set of transactions to include in each Rollup block.

B) What could go wrong if the malicious coordinator used the strategy you outlined in the previous part? Give an example of how Alice might lose some assets as a result. If needed, you can assume that there is an exchange (e.g., Uniswap) running in the Rollup that will exchange ETH for SOL and vice versa.

C) Can you think of a way to ensure that the coordinator always pushes the same Merkle root to both chains?