CS251: Cryptocurrencies and Blockchain Technologies **Fall 2025**

Assignment #4

Due: 11:59pm on Tue., **Dec. 2, 2025**
Submit via Gradescope (each answer on a separate page) code: **KDJ7NE**

Problem 1. (*Private payments*) In Lecture 15 we looked at how zero knowledge proofs can be used to provide privacy in a payment system like Tornado. In particular, on Slide 32 we defined the statement that needs to be proved in zero knowledge during withdrawal. As usual, for a statement x the prover is proving that it knows a witness w such that $Cir(x, w) = 0$, where Cir is an arithmetic circuit that checks the three conditions listed on the slide.

a. Suppose the circuit Cir did not check condition (iii) on Slide 32, that is, it did not verify that $\mathrm{nf} = H_2(k')$. What would go wrong in the system?

b. Suppose the proof system that is used to prove knowledge of w were not zero knowledge. For example, suppose that the proof π leaked the entire witness w. What would go wrong in the system?

Problem 2. (*Polynomial commitments*) In Lecture 16, starting on slide 14, we defined the concept of a polynomial commitment scheme (PCS). In this exercise we will develop an important application for a PCS. First, let us briefly review what is a PCS. A PCS is a tuple of four algorithms: *setup*, *commit*, *prove*, and *verify*. The PCS is initialized by running $setup(\lambda, d)$ to obtain some public parameters pp that lets one commit to polynomials of degree at most d. Here λ is the security parameter that determines the security level of the scheme: we typically set $\lambda = 128$. Carol (the committer) has a univariate polynomial $f \in \mathbb{F}_p[X]$ of degree at most d. Carol can commit to f by sending to Roger (the recipient) a commitment string com_f obtained by running $commit(pp, f)$. Later, Roger can choose some $u \in \mathbb{F}_p$ and ask Carol to send him $v := f(u) \in \mathbb{F}_p$ along with a proof $\pi_{u,v}$ that v is indeed the evaluation of the committed polynomial at u. Carol constructs the proof by running $\pi_{u,v} \leftarrow Prove(pp, (u, v), f)$. This $\pi_{u,v}$ is called an *evaluation proof*. Roger can verify the proof by running $verify\big(pp, (com_f, u, v), \pi_{u,v}\big)$ which outputs accept or reject. If *verify* outputs accept then Roger is convinced that (i) Carol has a polynomial $f \in \mathbb{F}_p[X]$ of degree at most d whose commitment is com_f, and (ii) this f satisfies $f(u) = v$. A PCS must satisfy several security properties that we will not discuss here (at the very least, the commitment must be binding). There are PCS constructions where com_f and $\pi_{u,v}$ are as short as 200 bytes each, no matter what d is.

Now, let's develop a cool application for a PCS. Suppose Carol has a set $S = \{s_1, \ldots, s_n\} \subseteq \mathbb{F}_p$. Carol wants to commit to S so that later, given some $s \in \mathbb{F}_p$, if s is in S then she can convince Roger of that fact (an inclusion proof), and if s is not in S then she can convince Roger of that fact (an exclusion proof). One solution is to commit to S using a Merkle tree, where the Merkle root is the commitment to S. Then, for $s \in S$ she can send Roger a Merkle proof of size $O(\log n)$ to convince Roger that s is in S. Let's show that we can do better using a PCS.

a. Show how Carol can use a PCS to commit to the set S so that later, when Roger sends an $s \in \mathbb{F}_p$, Carol can provide an inclusion or an exclusion proof for s, using a single evaluation

proof, that convinces Roger. Explain how Carol commits to S, and how she constructs the exclusion or inclusion proof for a given $s \in \mathbb{F}_p$.
Hint: consider having Carol use the polynomial $f_S(X) := (X - s_1) \cdots (X - s_n) \in \mathbb{F}_p[X]$.

b. For a large n, the inclusion/exclusion proofs in part (a) are already shorter than a Merkle proof. Let's do even better: let's build a batch inclusion proof — something that cannot be done with a Merkle tree. Suppose Roger sends to Carol distinct $u_1, \ldots, u_k \in \mathbb{F}_p$ and all of them happen to be in S. Carol wants to convince Roger of that fact. Using a Merkle tree, Carol would need to send over a proof of size $O(k \log n)$ — one Merkle inclusion proof for each u_i. Show that using the commitment scheme from part (a), Carol can convince Roger using a *constant size proof* (independent of n and k).

Describe the message flow in your interactive batch inclusion protocol between Carol and Roger. Make sure to explain why your batch inclusion protocol is complete (i.e., Roger will always accept a proof by an honest Carol) and sound (i.e., if one of $u_1, \ldots, u_k$ are not in S then no matter what Carol does, Roger will accept with negligible probability). You may assume that n/p is negligible. Note that your interactive protocol can be made non-interactive using the Fiat-Shamir transform on slide 9.
Hint: Both Carol and Roger can construct the polynomial $g(X) := (X - u_1) \cdots (X - u_k)$. Carol will then prove to Roger that $f_S(X)$ is a multiple of $g(X)$. That is, there exists a quotient polynomial $q \in \mathbb{F}_p[X]$ such that $f_S = g \times q$. Carol can send to Roger a commitment to q, and then prove that indeed $f_S = g \times q$ using the polynomial equality testing protocol from Lecture 16 slide 36.

Discussion: developing this further leads to a data structure called a *Verkle tree*, which has much shorter proofs than a Merkle tree.

Problem 3. (*An insecure 3-party payment channel*) Three parties, A, B, and C, are constantly making pairwise payments and thus design a 3-party Bitcoin payment channel based on the bidirectional payment channel we saw in Lecture 17. To establish the channel the three parties create a 3-out-of-3 multisig address that is bound to the public keys of A, B, and C, and all three send some initial funds to that address. Once the channel is established, they can transact without ever touching the blockchain. For example, when B wants to pay A using the channel, the following happens without touching the blockchain:

- Party B sends to A a hashed timelocked transaction T_{A} that is already signed by B and C. The transaction has three outputs:
  - one immediate output for B whose value is B's current balance in the channel,
  - one immediate output for C whose value is C's current balance in the channel, and
  - one output whose value is A's current balance in the channel, but with a hashed timelock spending rule: A can spend the output seven days after the transaction is posted, but either B or C can spend this output immediately if they have a hash preimage x initially known only to A.

  As in the two party payment channel, if A wants to close the channel she will sign this transaction T_A and post it. B and C will collect their balances immediately, and A will collect her balance after seven days. However, if A wants to keep using the channel, then when she later pays B, she would first send the preimage x to B and C, thereby effectively invalidating the transaction T_{A}. Indeed, it would no longer make sense for A to post this stale transaction T_A: if she did, then either B or C would immediately use x to spend A's timelocked output, and A would lose her balance in the channel. This means that she can no longer close the channel in its old pre-payment state. A would then obtain from B and C a new transaction T'_A (with a similar structure as T_A) that lets her close the channel in its new state, if she wants.

- Parties B and C each receive from their peers a similar transaction with three outputs representing the current balances in the channel. For example, party B's transaction has one output that is immediately available for A, one output that is immediately available for C, and one output that is hashed timelocked for B as above. C receives a similar transaction.

Let's show that while this general approach is secure for a two-party payment channel, it is completely insecure for three parties. In particular, two colluding parties can steal funds from the third. To see how, suppose the channel has a total of 100 BTC locked up. At some time in the past, 90 BTC belonged to A and 5 BTC belonged to B and C each. Currently 80 of the BTC belong to C and 10 BTC belong to A and B each. Show that A and B can collude to steal 75 BTC that currently belong to C, and split the loot between them. You may assume that A and B can post successive transactions before C can react.
Hint: think about what happens if A posts the stale transaction that lets her close the channel at the time when 90 of the BTC belonged to her.