COMS 4995-001 (Science of Blockchains): Homework #3

Due by 11:59 PM on Wednesday, February 19th, 2025

Instructions:

(1) Solutions are to be completed and submitted in pairs.

(2) We are using Gradescope for homework submissions. See the course home page for instructions, the late day policy, and the School of Engineering honor code.

(3) Please type your solutions if possible and we encourage you to use the LaTeX template provided on the Courseworks page.

(4) Write convincingly but not excessively. (We reserve the right to deduct points for egregiously bad or excessive writing.)

(5) Except where otherwise noted, you may refer to your lecture notes and the specific supplementary readings listed on the course Web page *only*.

(6) You are not permitted to look up solutions to these problems on the Web. You should cite any outside sources that you used. All words should be your own. Submissions that violate these guidelines will (at best) be given zero credit, and may be treated as honor code violations.

(7) You can discuss the problems verbally at a high level with other pairs. And of course, you are encouraged to contact the course staff (via the discussion forum or office hours) for additional help.

(8) If you discuss solution approaches with anyone outside of your pair, you must list their names on the front page of your write-up.

Throughout this homework assignment, unless otherwise specified, we consider a network of n validators, denoted by $\{v_1, \ldots, v_n\}$.

Problem 1

(10 points) This question concerns the Tendermint protocol from Lecture 6 (analyzed, as usual, in the partially synchronous model from Lecture 4). In the lecture, we proved that the Tendermint protocol is consistent (always) and live (post-GST) provided $f < \frac{n}{3}$ validators are Byzantine (with the other validators assumed non-faulty). But what if there are at least $\frac{n}{3}$ Byzantine validators? Is the Tendermint protocol still consistent? Provide either a proof of consistency or an explicit execution of the protocol that demonstrates a consistency violation.

Problem 2

This question also concerns the Tendermint protocol in partial synchrony.

1. (10 points) Suppose we wanted to modify the Tendermint protocol to have a stronger consistency guarantee, at the cost of a weaker liveness guarantee. Specifically, suppose we want a protocol that remains consistent (always) in the presence of at most $f < \frac{2n}{5}$ Byzantine faults, but is guaranteed to be live (post-GST) only in the presence of at most $f < \frac{3n}{10}$ Byzantine faults. What changes are required to the Tendermint protocol in order to achieved these revised guarantees? What changes to

the correctness proof (i.e., of consistency and post-GST liveness) are necessary in order to accommodate these protocol modifications?

2. (5 extra credit points) More generally, suppose we had two parameters, t_c and t_ℓ, and wish to design a protocol that is guaranteed to be consistent (always) with $\leq t_c$ Byzantine faults, and is guaranteed to be live (post-GST) with $\leq t_\ell$ Byzantine faults. Identify values of $\alpha, \beta > 0$ such that the following statement holds: such a protocol exists if $\alpha t_c + \beta t_\ell < n$, and such a protocol does not exist if $\alpha t_c + \beta t_\ell > n$. A complete solution should describe a correct protocol for all values of t_c, t_ℓ with $\alpha t_c + \beta t_\ell < n$, and a proof sketch (along the lines of that in Lecture 5) that no correct protocol exists for any values of t_c, t_ℓ with $\alpha t_c + \beta t_\ell > n$.

Problem 3.

(10 points) This problem also concerns the Tendermint protocol from Lecture 6. Exhibit an execution of the protocol in the partially synchronous setting with the following property: at some timestep t, there exist non-faulty validators i and j such the local chain A_i of validator i is inconsistent with the local chain C_j of validator j (i.e., neither of these chains is a prefix of the other). Your example should make use of strictly less than $n/3$ Byzantine validators.

Problem 4.

(15 points) Recall the permissioned version of longest-chain consensus from Lecture 7. Recall that a sequence $\ell_1, \ldots, \ell_m$ of leaders is ω-*balanced* if every window of at least ω leaders contains a strict majority of honest leaders. More formally, for every $1 \leq i, j \leq m$, if $i - j \geq \omega - 1$, then the set $\{\ell_i, \ldots, \ell_j\}$ contains a strict honest majority. For the purposes of the following questions, you should prove a tight bound on your answers—that is, you should both prove that the value you suggest is sufficient to satisfy the required condition, and show (by proof or explicit counterexample) that any smaller choice is insufficient.

1. Suppose leaders are sequenced in round-robin fashion, using repeated passes over an arbitrary (but fixed) ordering of the n validators. Assume that of those n validators, a strict minority are Byzantine. Find the smallest value of ω for which it is guaranteed that any round-robin leader sequence of the n validators is ω-balanced.

2. Suppose $f < \frac{n}{2}$. What is the smallest ω for which *there exists* a round-robin leader sequence that is ω-balanced?

3. Same as the previous subproblem, except under the stronger assumption that $f < \frac{n}{3}$.

Problem 5.

(20 points) This problem is a reading response problem. Specifically, you should read up on the CometBFT implementation of Tendermint (another useful link: Staking). Many blockchain protocols based on Tendermint manage the voting power of their validators (i.e., how much each validator contributes toward a read or write quorum) via a *staking* mechanism (as opposed to the "one validator one vote" approach that we used in lecture). Intuitively, a validator's power in the protocol is then proportional to its share of the overall financial stake in the protocol. In your response, discuss how CometBFT manages stake and the role that stake plays in their implementation of the Tendermint protocol. Explain what is unbonding time, what it is used for and why, and how long it is. Discuss the potential advantages and disadvantages of having a longer unbonding time vs. a shorter unbonding time.

[Target length: three paragraphs. The more specific, the better.]

Problem 6

(20 points)

Bitcoin Testnet4 Demo

In the second part of this assignment, we'll make a transaction on a Bitcoin testnet, like we did for Sepolia in HW 1.

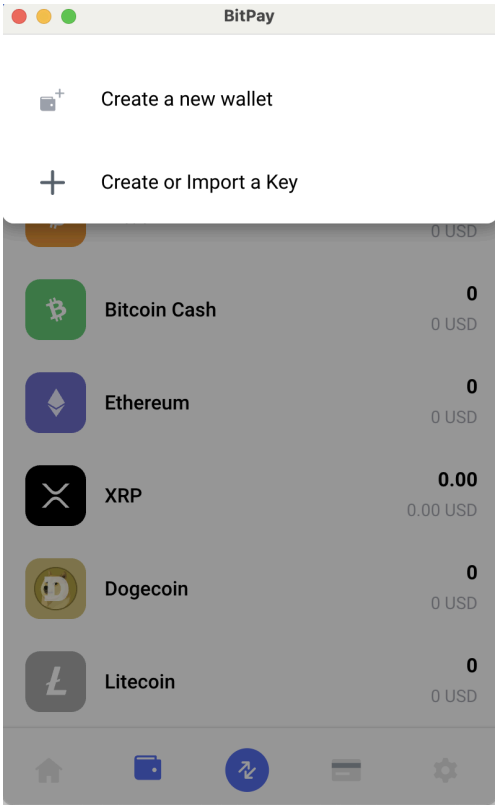
We'll primarily be following along these instructions:
https://support.bitpay.com/hc/en-us/articles/360015463612-How-to-Create-a-Testnet-Wallet

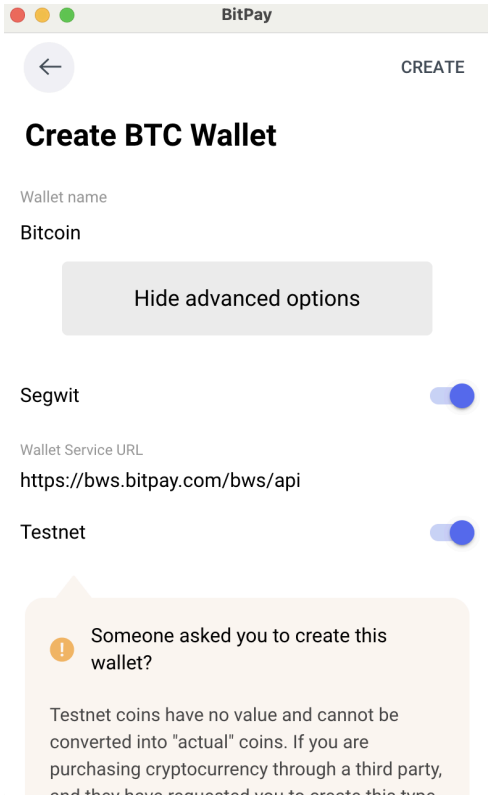
Step 1: Download BitPay

The wallet application we'll be using in this demo is called BitPay. We can't use MetaMask here because it doesn't support the Bitcoin Testnet. You can download BitPay here:
https://support.bitpay.com/hc/en-us/articles/360011332152-How-to-install-the-BitPay-Wallet

As before, we're using a Testnet, so your wallet will not hold any tokens of actual value. As such, feel free to skip through the recovery options, etc. unless you plan to use this wallet for other purposes in the future.

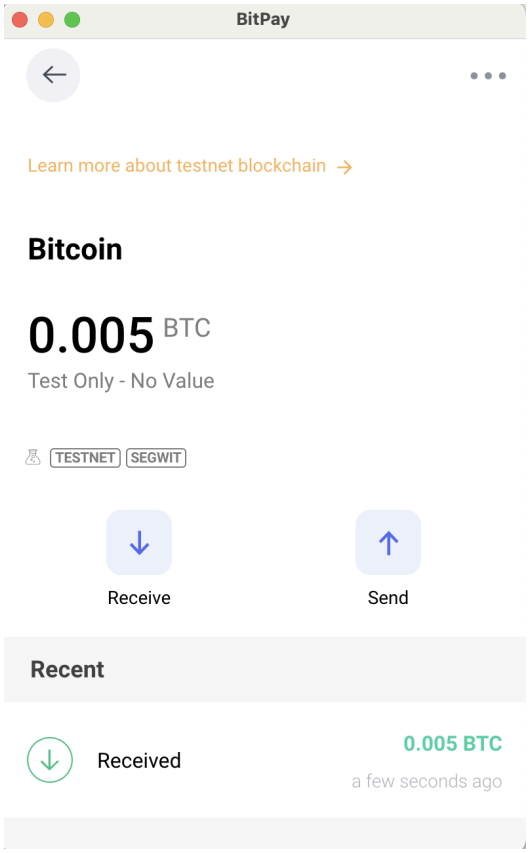
Step 2: Create a new Testnet wallet

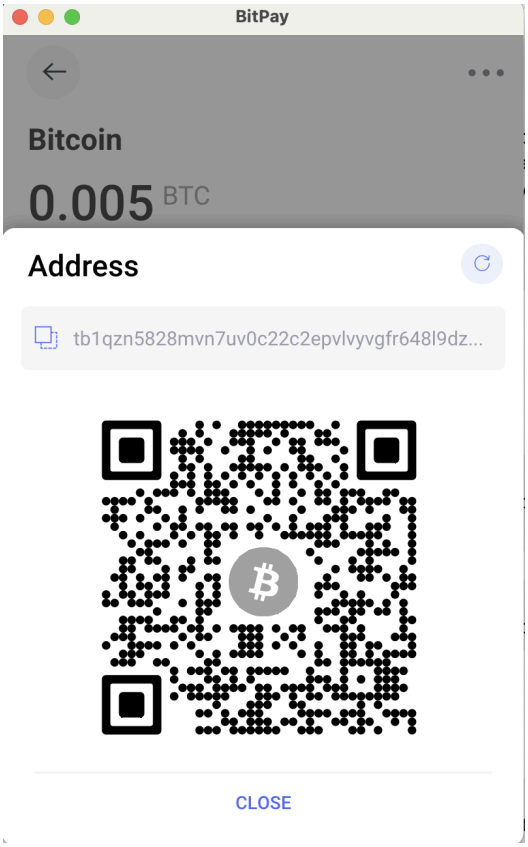




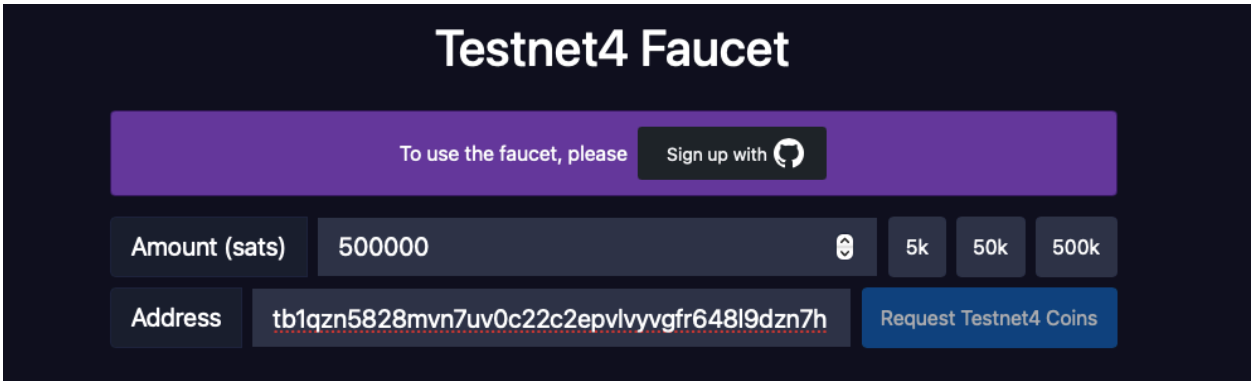
You can do this by clicking the three dots in the upper right on the wallet tab, and selecting “Create a new wallet”. Then, select “Simple Wallet” → Bitcoin (BTC) → “Show advanced options” → and toggle “Testnet” to on as in the second picture above. Then, click on “Create” in the upper right corner.

Step 3: Receive test BTC from faucet.





First, click on “Receive” in BitPay and copy your address. Unlike the images above, your wallet will show 0.000 BTC as its balance.

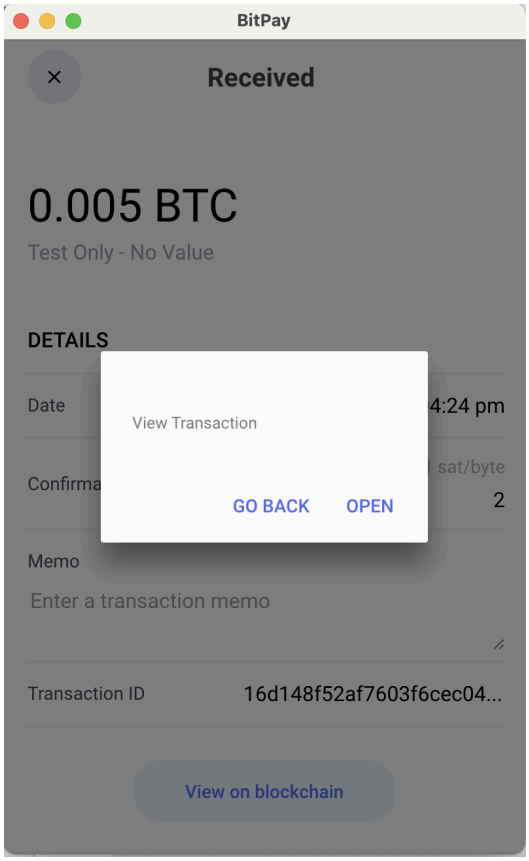


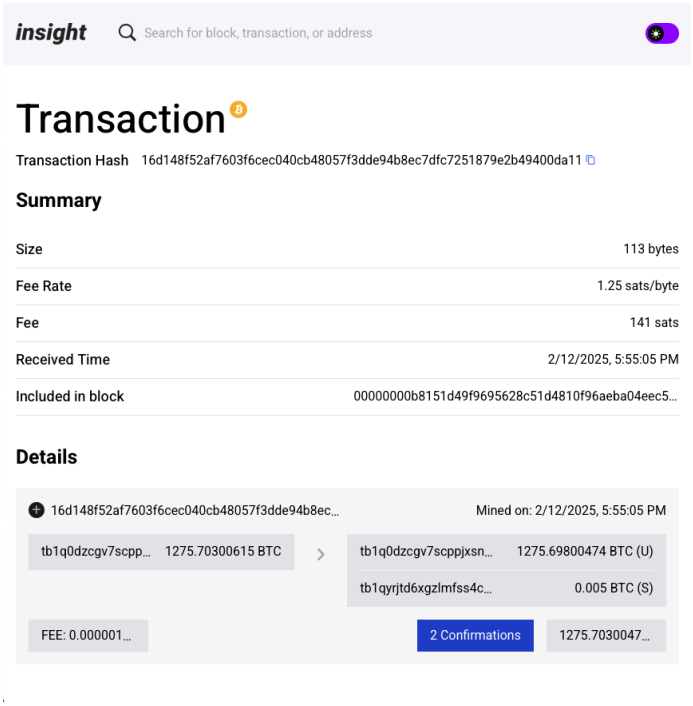
Next, go to https://mempool.space/testnet4/faucet

This is the faucet we'll be using. Sign in with a GitHub account to use it and paste in your address that you copied earlier. **Make sure to select 500k for the amount, or else you may not have enough funds to clear the network fees when we eventually spend the tBTC.**

Step 4: Wait to receive BTC

As discussed in class, it may take a bit of time for your transaction to finalize. You'll see this in your wallet as it transitions from saying "Receiving" to "Received". Click on the transaction (it should show up under "Recent") and click on "View on blockchain" to get information about the transaction. A page that looks similar to Etherscan from HW 1 will appear.

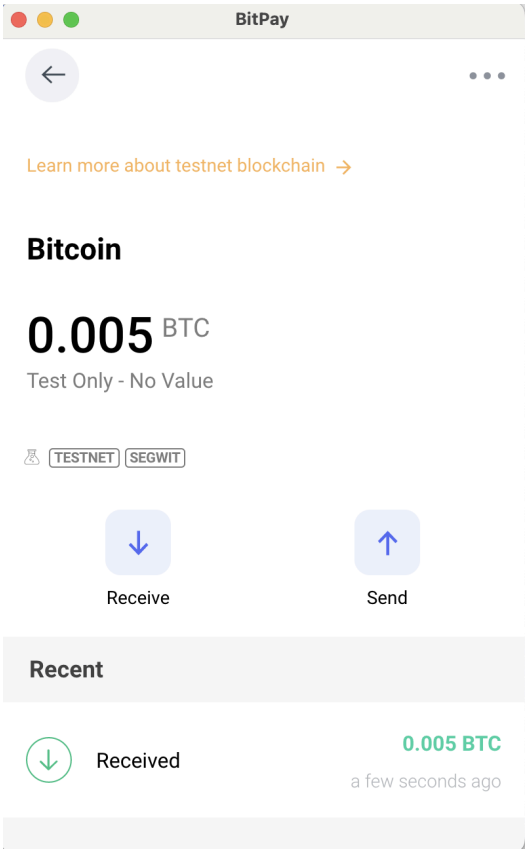


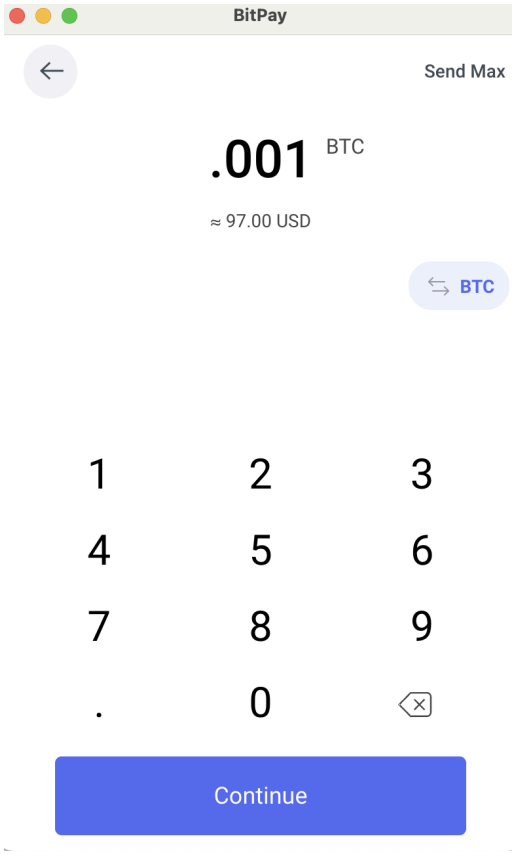


The URL for that page is one of the two things that you'll submit for this part of the assignment.

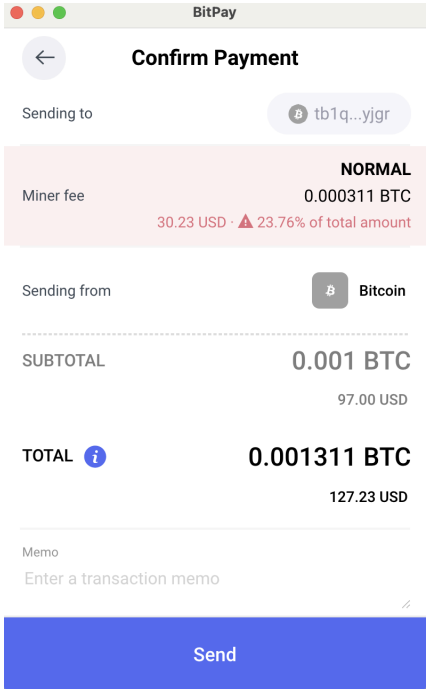
Step 5: Send BTC back to the wallet

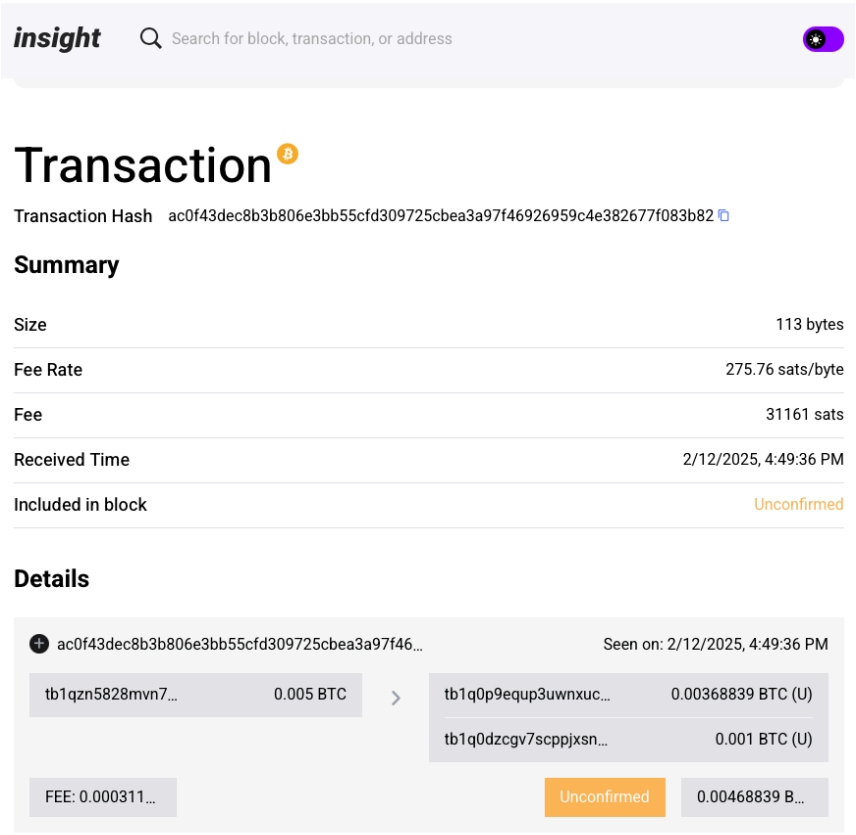
Now that we have BTC, we're going to make another transaction to send some of it back to the faucet that we just got it from. This time, however, we're going to have to pay the transaction fees since we will be making the payment. The faucet's address that we'll be sending the BTC back to is: **tb1q0dzcgv7scppjxsnwlzpkt02vlmc5rtr40wyjgr**





Enter in 0.001 BTC and press continue. Then, press "Send". Again, notice that you'll have to wait a bit until the transaction shows up as "Sent" due to Bitcoin's slower finality relative to Ethereum.





As before, select the “Sent”or “Sending” transaction under recent and click “View in Blockchain” to get a page that looks like the image on the right. If you click on it while the transaction still shows up as “Sending”, the page will look like the image on the right, where it shows that the transaction is still unconfirmed. Copy the URL for this page, as you will submit it for credit.

Step 6: Submission

In your PDF submission, include the links for the BitPay insight pages in steps 4 and 6. Congrats, you’ve made your first payment on a Bitcoin testnet!

Write a couple of sentences comparing and contrasting your experience here versus in the Sepolia demo. Can you explain any of the differences using what we’ve learned in class?