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

Assignment #1

Due: 11:59pm on **Tue, Oct. 7, 2025** at 11:59pm
Submit via Gradescope (each answer on a separate page) code: **KDJ7NE**

Problem 1. (*A broken proof of work hash function*) In class we discussed using a hash function $H : X \times Y \to \{0, 1, \ldots, 2^n - 1\}$ for a proof of work scheme. Once an $x \in X$ and a difficulty level D are published, it should take an expected D evaluations of the hash function to find a $y \in Y$ such that $H(x, y) < 2^n/D$. Suppose that $X = Y = \{0, 1\}^m$ for some m (say $m = 512$), and consider the hash function

$$H : X \times Y \to \{0, 1, \ldots, 2^{256} - 1\} \qquad \text{defined as } H(x, y) := \text{SHA256}(x \oplus y).$$

Here $\oplus$ denotes a bit-wise xor. Show that this H is insecure as a proof of work hash. In particular, suppose D is fixed ahead of time. Show that a clever attacker can find a solution $y \in Y$ with minimal effort once $x \in X$ is published. Hint: the attacker will do most of the work before x is published.

Answer: The hash function $H(x, y) = \text{SHA256}(x \oplus y)$ is insecure because an attacker can do all the work before x is known and then instantly find a valid solution.
Attack: Before x is published, the attacker tests random values $y_1, y_2, y_3, \ldots \in Y$ until finding a y such that $\text{SHA256}(y) < 2^{256}/D$. This takes about D hash evaluations, and the attacker saves this y.
After x is revealed, the attacker computes $y' = x \oplus y$ and submits y' as the solution.
To verify: $H(x, y') = \text{SHA256}(x \oplus y') = \text{SHA256}(x \oplus (x \oplus y)) = \text{SHA256}(y) < 2^{256}/D$, so the solution is valid.
This means the attacker can precompute all work before x is published, breaking the proof-of-work requirement that effort must be done after the challenge is known.

Problem 2. (*Beyond binary Merkle trees*) Alice can use a binary Merkle tree to commit to a list of elements $S = (T_1, \ldots, T_n)$ so that later she can prove to Bob that $S[i] = T_i$ using an inclusion proof containing at most $\lceil \log_2 n \rceil$ hash values. The binding commitment to S is a single hash value. In this question your goal is to explain how to do the same using a k-ary tree, that is, where every non-leaf node has up to k children. The hash value for every non-leaf node is computed as the hash of the concatenation of the values of all its children.

a. Suppose $S = (T_1, \ldots, T_9)$. Explain how Alice computes a commitment to S using a ternary Merkle tree (i.e., $k = 3$). How does Alice later prove to Bob that T_4 is in S? What values are provided in the proof?

Answer: The tree consutrction can look something like this: Alice builds a ternary tree with leaves $T_1, \ldots, T_9$. At level 1, she computes $H_{1,0} = H(T_1 \parallel T_2 \parallel T_3)$, $H_{1,1} = H(T_4 \parallel T_5 \parallel T_6)$, and $H_{1,2} = H(T_7 \parallel T_8 \parallel T_9)$. At level 2 which is the root, she computes $H_{2,0} = H(H_{1,0} \parallel H_{1,1} \parallel H_{1,2})$. The commitment is the root hash $H_{2,0}$.
Proof that $T_4 \in S$: Alice provides (1) the value T_4, (2) sibling values T_5 and T_6, and (3) sibling hashes $H_{1,0}$ and $H_{1,2}$.
Verification: Bob computes $H_{1,1} = H(T_4 \parallel T_5 \parallel T_6)$, then computes the root as $H(H_{1,0} \parallel H_{1,1} \parallel H_{1,2})$, and checks this equals the committed value $H_{2,0}$.

b. Suppose S contains n elements. What is the length of the proof that proves that $S[i] = T_i$, as a function of n and k?

Answer: The proof length is $(k-1) \cdot \lceil \log_k n \rceil$ hash values (plus the element T_i itself). The tree has height $h = \lceil \log_k n \rceil$. At each level along the path from leaf to root, we need to provide the $(k-1)$ sibling hashes so therefore, the total is $(k-1) \cdot h = (k-1) \cdot \lceil \log_k n \rceil$ hashes.

c. For large n, if we want to minimize the proof size, is it better to use a binary or a ternary tree? Why?

Answer: Binary trees are better for minimizing proof size.
For binary ($k = 2$): proof size $= \lceil \log_2 n \rceil$ hashes.
For ternary ($k = 3$): proof size $= 2\lceil \log_3 n \rceil = 2\lceil \frac{\log_2 n}{\log_2 3} \rceil \approx 2\lceil \frac{\log_2 n}{1.585} \rceil \approx 1.26\lceil \log_2 n \rceil$ hashes.
As k increases, the tree height decreases which is good but siblings per level increases which is bad. The proof size $(k-1) \cdot \log_k n = (k-1) \cdot \frac{\ln n}{\ln k}$ is minimized when $k = e \approx 2.718$. Since k must be an integer, comparing $k = 2$ (proof $\approx \log_2 n$) versus $k = 3$ (proof $\approx 1.26 \log_2 n$) shows that $k = 2$ gives the smallest proof.

Problem 3. (*Bitcoin script*) Alice is on a backpacking trip and is worried about her devices containing private keys getting stolen. She wants to store her bitcoins in such a way that they can be redeemed via knowledge of a password P. Accordingly, she stores them in the following `ScriptPubKey` address:

```
OP_SHA256
<0xeb271cbcc2340d0b0e6212903e29f22e578ff69b>
OP_EQUAL
```

where the value `0xeb271...` is the hex encoding of SHA256(P).

a. Write a `ScriptSig` script that will successfully redeem this UTXO given the password P.
Hint: it should only be one line long.

Answer:

`<P>`

The `ScriptSig` pushes `P` onto the stack. `OP_SHA256` hashes it, the stored hash constant is pushed, and `OP_EQUAL` checks they match. If they do, then script succeeds.

b. Suppose Alice chooses a six character password P. Explain why her bitcoins can be stolen soon after her UTXO is posted to the blockchain. You may assume that computing SHA256 of all six character passwords can be done in reasonable time.

Answer: Once the UTXO is posted with SHA256(P) = `0xeb271...`, an attacker can recover P by brute force. There are about $62^6 \approx 57$ billion six-character alphanumeric passwords, and modern hardware can try that many SHA256 hashes in minutes, so the attacker can find P and spend the coins by creating a transaction with `ScriptSig: <P>` which will send the funds to their address.
Even if the attacker doesn't precompute P, Alice faces another risk which is when she broadcasts her transaction containing `<P>`, the password is visible in the mempool. An attacker watching the mempool can copy `<P>` into a new transaction that pays a higher fee; miners will prefer the higher-fee transaction, so the attacker's transaction gets mined first and Alice's fails.

c. Suppose Alice chooses a strong 30 character passphrase P. Is the `ScriptPubKey` above a secure way to protect her bitcoins? Why or why not?
Hint: reason through what happens when she tries to redeem her bitcoins.

Answer: No, this setup isn't secure even if Alice uses a strong password. When she redeems her bitcoins, she has to publish a transaction with `ScriptSig: <P>`, which exposes the password `P` to everyone watching the mempool before it's confirmed. An attacker can see `P`, quickly copy it, and create a new transaction using the same password but sending the coins to their own wallet, while offering a higher fee. Miners will choose the attacker's transaction since it pays more, and Alice's will fail. The main issue is that the password has to be revealed in the script to unlock the coins, creating a race. A safer design would use `OP_CHECKSIG` and public-key cryptography, where Alice proves ownership of her private key with a unique digital signature that can't be reused by others.

Problem 4. (*BitcoinLotto*) Suppose the nation of Bitcoinia decides to convert its national lottery to use Bitcoin. A trusted scratch-off ticket printing factory exists and will not keep records of any values printed. Bitcoinia proposes a simple design: a weekly public address is published that holds the jackpot. This allows everyone to verify that the jackpot exists, namely there is a UTXO bound to that address that holds the jackpot amount. Then a weekly run of tickets is printed so that the winning lottery ticket contains the correct private key under the scratch material. Other tickets contain a random string under the scratch material.

a. If the winner finds the ticket on Monday and immediately claims the jackpot, this will be bad for sales because players will all realize the lottery has been won. Modify the design to use one of the locktime opcodes to ensure that the prize can only be claimed at the end of the week. (of course, you cannot prevent the winner from proving ownership of the correct private key outside of Bitcoin). The Bitcoin script opcodes are listed at `https://en.bitcoin.it/wiki/Script`. Alternatively, explain how to use a 2-out-of-2 multisig for this. You only need to write up one design: either using locktime or multisig.

Answer: Solution using `OP_CHECKLOCKTIMEVERIFY`:
`ScriptPubKey` for the weekly jackpot:

```
<end_of_week_timestamp> OP_CHECKLOCKTIMEVERIFY OP_DROP
<winner_pubkey> OP_CHECKSIG
```

The lottery publishes `winner_pubkey` at the start of the week, and the winning ticket contains the corresponding private key. The winner creates a signed transaction spending the UTXO but sets the transaction's `nLockTime` field to `end_of_week_timestamp`. The transaction is only valid in blocks with timestamp $\geq$ `end_of_week_timestamp`. If broadcast early, nodes reject it. At week's end, the transaction becomes valid and can be mined. The winner can prove ownership off-chain by signing messages, but cannot claim on-chain until the locktime expires.

b. Some tickets inevitably get lost or destroyed. Modify the design to roll forward so that any unclaimed jackpot from Week n can be claimed by the winner in Week $n+1$. If the Week $n+1$ jackpot is unclaimed, then the jackpot from both weeks n and $n+1$ can be claimed by the winner of Week $n+2$, and so on. Can you propose a design that works without introducing new ways for the lottery administrators to embezzle funds beyond what is possible on the basic scheme described at the beginning of the question? You may assume that the lottery system runs for 1000 weeks, and then shuts down permanently. Hint: use multisig.

Answer: Design using pre-signed transaction chain with 2-of-2 multisig:
At initialization, generate keypairs: (sk_n, pk_n) for each week n winner, master rollover keys (msk_n, mpk_n) for $n = 2, \ldots, 1000$, and charity key (csk, cpk) for final unclaimed funds.
Week n `ScriptPubKey: 2 <pk_n> <mpk_{n+1}> 2 OP_CHECKMULTISIG`
At setup, create and publish all 1000 rollover transactions. Rollover transaction for Week $n \to n+1$ has input from Week n UTXO, output to Week $n+1$ address (combining Week n + Week $n+1$ amounts), locktime at end of Week $n+1$, and is pre-signed with msk_{n+1} but still needs signature from sk_n.
If Week n winner claims during Week $n+1$, they sign with sk_n and lottery provides msk_{n+1} signature after Week n ends. If unclaimed, anyone broadcasts the pre-signed rollover at Week $n+1$ end, moving funds to Week $n+1$ UTXO. Week $n+1$ winner then claims combined jackpot. All rollover transactions are published at initialization, preventing the lottery from redirecting funds. After Week 1000, unclaimed funds go to the pre-specified charity.

Problem 5. (*Lightweight clients*) Suppose Bob runs an ultra lightweight Bitcoin client which receives the current head of the blockchain from a trusted source. This client has very limited memory and so it only stores the block header of the most recent block in the chain (the head of the chain), deleting any previous block headers.

a. Consider the current blockchain design, where every block header contains (i) a Merkle root of all the transaction in the block, and (ii) the hash of the previous block header. Suppose Alice posts a transaction on chain that sends a payment to Bob. Some time later Alice want to prove to Bob that she paid him. Bob runs an ultra lightweight client, and only has the latest block header. What information should Alice send to Bob to prove that her payment to Bob has been included in a block on chain?

Answer: Alice must provide: (1) The transaction itself showing the payment to Bob. (2) A Merkle proof consisting of sibling hash values along the path from the transaction's position to the Merkle root, proving the transaction is in a specific block. (3) A chain of block headers from the block containing her transaction up to the current head, where each header contains the hash of the previous header.
Bob verifies by: (a) Using the Merkle proof with the transaction to compute the Merkle root and checking it matches the root in the claimed block header. (b) Verifying each header in the chain correctly hashes to the next header's previous hash field. (c) Checking the final header matches his current head. This proves the transaction is in the main blockchain at the claimed depth.

b. Suppose Alice's payment was included in a block k blocks before the current head and there are exactly n transactions per block. Estimate how many bytes this proof will require in terms of n and k. Compute the proof size for $k = 6$ and $n = 1024$, assuming SHA256 is used as the hash function throughout.

Answer: Proof components: (1) Transaction: approximately 250 bytes. (2) Merkle proof: the tree has height $\lceil \log_2 n \rceil$, requiring one 32-byte hash per level, totaling $32\lceil \log_2 n \rceil$ bytes. (3) Block header chain: k headers of 80 bytes each (Bitcoin header format: 4+32+32+4+4+4 = 80 bytes), totaling $80k$ bytes.
Total: $\approx 250 + 32\lceil \log_2 n \rceil + 80k$ bytes.
For $k = 6$ and $n = 1024$: Transaction = 250 bytes, Merkle proof = $32 \times \lceil \log_2 1024 \rceil = 32 \times 10 = 320$ bytes, headers = $80 \times 6 = 480$ bytes. **Total = 1050 bytes $\approx$ 1 KB.**