**Problem 1. Hash functions and proofs of work.** In class we defined two security properties for a hash function, one called collision resistance and the other called proof-of-work security. Show that a collision-resistant hash function may not be proof-of-work secure.

**Hint:** let $H : X \times Y \to \{0, 1, \ldots, 2^n - 1\}$ be a collision-resistant hash function. Construct a new hash function $H' : X \times Y \to \{0, 1, \ldots, 2^m - 1\}$ (where $m$ may be greater than $n$) that is also collision resistant, but for a fixed difficulty $D$ (say, $D = 2^{32}$) is not proof-of-work secure with difficulty $D$. That is, for every puzzle $x \in X$ it should be trivial to find a solution $y \in Y$ such that $H'(x, y) < 2^m/D$. This is despite $H'$ being collision resistant. Remember to explain why your $H'$ is collision resistant, that is, explain why a collision on $H'$ would yield a collision on $H$.

**Problem 2. Beyond binary Merkle trees.** Alice can use a binary Merkle tree to commit to a set of elements $S = \{T_1, \ldots, T_n\}$ so that later she can prove to Bob that some $T_i$ is in $S$ using a 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 $S$?

**b.** Suppose $S$ contains $n$ elements. What is the length of the proof that proves that some $T_i$ is in $S$, as a function of $n$ and $k$?

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

**Problem 3. Nakamoto consensus.** Which of the following does Nakamoto consensus guarantee (or guarantee with all but negligible probability)?

(a) agreement

(b) validity

(c) termination

(d) fault-tolerance

**Explain your answer.**

**Problem 4. Byzantine generals.** Can you solve the Byzantine generals problem (in a synchronous setting) with three generals one of whom may be a traitor? If so, how? If not, why not?

**Problem 5.** Consider a newspaper-classified-ad based cryptocurrency blockchain in which coins look like this:

$$\begin{aligned} \text{coin}_1 &:= \{\text{Create \$1 (serial\#53401) for } PK_{\text{fed}}\}_{SK_{\text{fed}}} \\ \text{coin}_2 &:= \{\text{Pay SHA256}(\text{coin}_1) \text{ to } PK_{\text{david}}\}_{SK_{\text{fed}}} \\ \text{coin}_3 &:= \{\text{Pay SHA256}(\text{coin}_2) \text{ to } PK_{\text{dan}}\}_{SK_{\text{david}}} \end{aligned}$$

Here the notation $\{m\}_{SK}$ denotes a pair $(m, \sigma)$ where $\sigma$ is a signature on $m$ generated using the key $SK$.

Every day the paper publishes a classified ad that looks like:

$$\text{new-coins, SHA256(yesterday's ad)}$$

Everyone using the system builds a database containing SHA256 of every coin ever published, along with one bit saying if the coin has already been spent. Nodes ignore blocks that contain a coin whose hash is already in the database (a duplicate hash) or spending a coin containing a spent or non-existent hash. For instance, an attempt to republish coin1 would fail, as would an attempt to re-spend coin2 such as:

$$\text{coin}_4 := \{\text{Pay SHA256}(\text{coin}_2) \text{ to } PK_{\text{david}}\}_{SK_{\text{david}}}$$

How might somebody attack this scheme if we use a malleable digital signature algorithm such as ECSDA? Recall that a signature scheme is said be malleable if for all messages $m$, a valid signature $\sigma$ on $m$ can be easily transformed into a different valid signature $\sigma'$ on $m$.

**Problem 6. 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. Accordingly, she stores them in the following `ScriptPubKey` address:

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

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

**b.** Suppose Alice chooses an eight character password. Explain why her bitcoins can be stolen soon after her UTXOs are posted to the blockchain. You may assume that computing SHA256 of all eight character passwords can be done in reasonable time.

**c.** Suppose Alice chooses a strong 20 character passphrase. 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.