EE 374: Blockchain Foundations **Stanford, Winter 2023**

Theory Exercise 3

Due: 1:15pm, Thursday, Mar 16, 2023

Please solve all the problems below. The problems are worth equal points. After a genuine attempt to solve the homework problems by yourself, you are free to collaborate with your fellow students to find solutions to the theory homework problems. Regardless of whether you collaborate with other students, you are required to type up or write your own solutions. Copying homework solutions from another student or from existing solutions is a serious violation of the honor code. Please take advantage of the instructors' and TA's office hours. We are here to help you learn, and it never hurts to ask! The assignments should be submitted via Gradescope.

Problem 1

In class, we studied *binary* Merkle trees, but Merkle trees can have higher dimensions. A d-dimensional Merkle tree is a *complete* tree in which every non-leaf node has exactly d children. The value of a leaf is the hash of its contents, as in a binary Merkle tree. The value of an internal node is the hash of the concatenation of its children's values. For example, in a ternary Merkle tree constructed using the hash function H, an internal node with value v which has children with values v_1, v_2, and v_3 will take the value $v = H(v_1 \,\|\, v_2 \,\|\, v_3)$.

Consider a ternary tree constructed using `blake2s`, a hash function with an output of $\kappa = 256$ bits which has 3^7 leaves. Calculate the proof size, in bytes, for this Merkle tree. You can assume the verifier already has the Merkle tree root and has received the claimed contents and index of the leaf node in question, so you do not need to include these in your proof size calculation.

The proof will have to contain all the siblings, at every level, of the path connecting the leaf to the root. Since the tree is ternary, there are two siblings at every level. The number of levels is $\log_3(3^7) = 7$. As we are using blake2s, each node contains a value $\kappa = 256$ bits long. The position of each child along the path can be determined by the index of the leaf, which is already known to the verifier. Therefore, the total proof size will be $2 \times 7 \times 256 = 3584$ bits $= 448$ bytes long.

Problem 2

In class, we studied Merkle trees where the leaves are relatively few. Many modern blockchain applications make use of *sparse* Merkle trees. A *sparse* Merkle tree is a complete binary Merkle tree with exactly 2^κ leaves. Since the tree has an exponential number of leaves, they cannot be processed by a polynomially bound computer. However, the vast majority of these leaves have contents set to the empty string, whereas only a polynomial number of leaves have non-empty contents. This makes computing the root and proofs within these trees efficiently possible.

Consider a *sparse* Merkle tree with 2^κ leaves constructed using `sha256`, a hash function with an output of $\kappa = 256$ in which the *first* leaf has contents equal to the string `hello` and value equal to

the hash

`2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824`.

All the other $2^{256} - 1$ leaves of this tree have their contents set to the empty string and their value is equal to

`e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`.

What is the value of the root of this tree?

We cannot calculate the value of every node individually, since there is an exponential number of them. Instead, we will approach this problem by calculating the value of the root of a tree where every leaf is empty. We will do this with trees that have height $1, 2, 3, \ldots, 256$. This can be done efficiently because the value of the root of a tree with height ℓ can be computed by hashing together the roots of the trees with height $\ell + 1$. After this has been done, we can keep hashing the non-empty value on the left-hand side with values of trees of empty leaves gradually increasing in height on the right-hand side.
The Python code to do this is depicted below.

```python
import hashlib

kappa = 256

def H(x):
  m = hashlib.sha256()
  m.update(x.encode('utf-8'))
  return m.hexdigest()

def sparse_root():
  y = H('hello')
  tmp = H('')
  for _ in range(kappa):
    y = H(y + tmp)
    tmp = H(tmp + tmp)
  return y

print(sparse_root())
```

Here is the equivalent code in JavaScript:

```javascript
const { createHash } = require('crypto');

function H(message) {
  const hash = createHash('sha256');
  hash.update(message);
  return hash.digest('hex');
}

let y = H('hello');
let tmp = H('');
for (let i = 0; i < 256; i++) {
  y = H(y + tmp);
```

```
  tmp = H(tmp + tmp);
}
console.log(y);
```

The final hash computed by the above code is:

7bff9d8451aeda8f9152b4bcc53b0472d09aec4fb627db15d00bdac49204092e.

Problem 3

Consider a *bitcoin backbone execution* with $n = 10$, $t = 2$, $q = 1000$, $\kappa = 256$ and $T = 2^{\kappa-19}$. Numerically calculate the following quantities:

1. The honest advantage δ.
2. The probability p of a successful query.
3. The probability f that a round is successful.
4. The expectation $\mathbb{E}[Z_r]$ of the number of successful adversarial queries in a round.
5. The expectation $\mathbb{E}[Z(S)]$ of the number of successful adversarial queries in a set of consecutive rounds with $|S| = 256$.
6. Your best lower bound for $\mathbb{E}[Y_r]$, the expectation of a round being a convergence opportunity.

1. From the honest majority equation $t < (1-\delta)(n-t)$ we obtain $2 < (1-\delta)8$. A threshold that captures this is therefore $\delta = 0.7$.
2. $p = \frac{T}{2^\kappa} = \frac{2^{\kappa-19}}{2^\kappa} = 2^{-19}$.
3. $f = 1-(1-p)^{q(n-t)} = 1-(1-2^{-19})^{1000\times 8} = 0.01514$.
4. $\mathbb{E}[Z_r] = pqt = 0.003815$.
5. $\mathbb{E}[Z(S)] = |S|\mathbb{E}[Z_r] = 256 \times 0.003815 = 0.9766$
6. $\mathbb{E}[Y_r] \geq q(n-t)p(1-p)^{q(n-t)-1} = 0.015$

Problem 4

Consider *bitcoin backbone executions* with $n = 10$, $t = 2$, $q = 1000$ and a hash function with $\kappa = 256$ bits. The probability of an execution of interest being typical is at least

$$1 - 2^{-(\epsilon^2 \lambda f/3 + \kappa - 30)}.$$

Numerically calculate:

1. The honest advantage δ.
2. Your choice of a secure parametrization for the probability f of a successful round and the Chernoff error ϵ that respects the balancing equation.
3. The probability of a successful query p.
4. A suitable target T to achieve it.
5. A Chernoff interval parameter λ that ensures typical executions occur with probability at least $1 - 2^{-256}$.
6. The chain growth parameters: The chain growth interval s and the chain velocity τ.
7. The common prefix parameter k.
8. The chain quality parameters: The chunk size ℓ and the chain quality μ.

1. As before, $\delta = 0.7$.
2. Using $\epsilon = f = \frac{\delta}{6}$, we get $\epsilon = f = 0.11$ (we round down to ensure the balancing equation remains satisfied).
3. Solving $f = 1 - (1 - p)^{q(n-t)}$ for p to obtain $p = 1 - (1 - f)^{1/q(n-t)}$, we calculate $p = 0.0000146$. This can also be calculated by plugging various p in the f expression and binary searching.
4. Solving $p = \frac{T}{2^\kappa}$ we obtain $T = 2^{\kappa-16}$.
5. Setting $\epsilon^2 \lambda f/3 + \kappa - 30 = 256$ and solving for λ we obtain $\lambda = 67618$.
6. The interval is $s = \lambda = 67618$ and the minimum velocity is $\tau = (1-\epsilon)f = (1-0.11) \times 0.11 = 0.0979$ blocks per unit of time.
7. The common prefix parameter is $k = 2\lambda f = 2 \times 67618 \times 0.11 = 14876$.
8. The chunk is $\ell = k = 2\lambda f = 14876$ and the minimum quality is $\mu = 1 - (1 + \frac{\delta}{2})\frac{t}{n-t} - \frac{\epsilon}{1-\epsilon} = 0.5389$.

We note that the Bitcoin backbone parametrization is quite conservative.

Problem 5

When we implemented the Selfish Mining simulation in class using the Monte Carlo method, we implicitly assumed that each block is instantly received by the whole network before the next block is mined. This allowed us to deduce that the probability of the next produced block being adversarial was $\frac{t}{n}$. However, when we explored the fan-out attack and formalized the bitcoin backbone model, we saw that time discretization into rounds leads to situations in which multiple successful queries can occur in a given round.

Consider a bitcoin backbone execution in the usual synchronous lockstep model. We call a round *adversarially superior* if the round contains an adversarially successful query, but is not a successful round (i.e., there is no honestly successful query). We are interested in the following event E:

An adversarially superior round has occurred *strictly prior* to the first successful round.

For example, if rounds 1, 2, and 3 contain no successful queries, round 4 contains one adversarially successful query, round 5 contains no successful queries, and round 6 contains one honestly successful query, then the event E has occurred. On the contrary, if the rounds 1, 2, and 3 contain no successful queries, and round 4 contains both an adversarial and an honestly successful query, the event E has not occurred, because the adversarially superior round was the same as the first successful round.

For simplicity, assume the execution continues for infinite, not just polynomial, duration.

1. Analytically calculate the probability of the event E occurring. Simplify your calculation to closed form (containing no summations $\sum$).
2. Use Bernoulli's approximation $(1-x)^a \approx 1-ax$ for large a and small x to simplify your expression.
3. Find the limit of your probability for $p \to 0$, i.e., successful queries become rarer and rarer.

1. Consider what it takes for the event E to be attained: A particular round i must be the first adversarially superior round and it must occur prior to any successful round (recall that a *successful round* means honestly successful). For the round i to be the first adversarially superior round, all of the following conditions must hold:
   a) All $(i-1)tq$ adversarial queries in the previous rounds must have failed, each with probability $1-p$. Otherwise round i would not be the *first* round in which the adversary succeeded. This happens with probability $(1-p)^{(i-1)tq}$.
   b) In order for the adversarial round to occur *strictly prior* to the first successful round, each of the $i(n-t)q$ honest queries in the previous and the current round must have failed, each with probability $1-p$. This happens with probability $(1-p)^{(i+1)(n-t)q}$.
   c) In order for the i^{th} round to be *adversarially superior*, it must be the case that *at least one* adversarial query among the tq in this round succeeded, i.e., *not all* tq queries in the round have been *unsuccessful*. This happens with probability $(1-(1-p)^{tq})$.

   These three conditions are independent, so we can multiply them together to get the probability that the i^{th} round has the desired property:

$$(1-p)^{itq}(1-p)^{(i+1)(n-t)q}(1-(1-p)^{tq})$$

   The event can occur in rounds with indices $i = 0 \ldots \infty$. These alternatives are mutually exclusive (it's not possible that this event happens in *both* round $i = 3$ *and* $i = 5$ – one of them must be the first), so we can sum them. Overall,

$$
\begin{aligned}
\Pr[E] &= \sum_{i=o}^{\infty} (1-p)^{itq}(1-p)^{(i+1)(n-t)q}(1-(1-p)^{tq}) \\
&= \sum_{i=0}^{\infty} (1-p)^{itq+(i+1)(n-t)q}(1-(1-p)^{tq}) \\
&= (1-(1-p)^{tq}) \sum_{i=0}^{\infty} (1-p)^{itq+(i+1)(n-t)q} \\
&= (1-(1-p)^{tq}) \sum_{i=0}^{\infty} (1-p)^{inq+(n-t)q} \\
&= (1-(1-p)^{tq})(1-p)^{(n-t)q} \sum_{i=0}^{\infty} ((1-p)^{nq})^i \\
&= (1-(1-p)^{tq})(1-p)^{(n-t)q} \left(\frac{1}{1-(1-p)^{nq}} \right)
\end{aligned}
$$

As a second method, this last probability can also be calculated directly using a different intuition: There's some round that contains the first successful query (whether honest or adversarial). We wish to calculate the probability that *that* round contains only adversarially successful queries and no honestly successful queries, conditioned on the event that it is successful:

$$
\begin{aligned}
\Pr[E] &= \Pr[r \text{ adversarially superior} | r \text{ has a successful query}] \\
&= \frac{\Pr[r \text{ adversarially superior } \textit{and} \text{ has successful query}]}{\Pr[r \text{ has successful query}]} \\
&= \frac{(1-(1-p)^{tq})(1-p)^{(n-t)q}}{1-(1-p)^{nq}}
\end{aligned}
$$

This is exactly the final expression we derived.

The way we simplified the $\sum$ was by using the geometric series formula.

As a third method, this last probability can be calculated by noticing that:

$$
\begin{aligned}
\Pr[E] &= \Pr[r \text{ adversarial } \textit{or } (r \text{ unsuccessful } \textit{and} \text{ future round adversarially superior})] \\
&= \Pr[r \text{ adversarial}] + \Pr[r \text{ unsuccessful } \textit{and} \text{ future round adversarially superior}] \\
&= \Pr[r \text{ adversarial}] + \Pr[r \text{ unsuccessful}] \cdot \Pr[E]
\end{aligned}
$$

Rearranging this equation:

$$\begin{aligned}\Pr[E] &= \frac{\Pr[r \text{ adversarial}]}{1 - \Pr[r \text{ unsuccessful}]} \\ &= \frac{\Pr[r \text{ adversarial}]}{\Pr[r \text{ has a successful query}]} \\ &= \frac{\Pr[r \text{ has a successful adversarial query}]\Pr[r \text{ has no successful honest query}]}{\Pr[r \text{ has a successful query}]} \\ &= \frac{(1-(1-p)^{tq})(1-p)^{(n-t)q}}{1-(1-p)^{nq}}\end{aligned}$$

This produces the same expression once more.

2.

$$\begin{aligned}\Pr[E] &\approx (1-(1-tqp))(1-(n-t)qp)\left(\frac{1}{1-(1-nqp)}\right) \\ &= tqp(1-(n-t)qp)\frac{1}{nqp} \\ &= \frac{t}{n}(1-(n-t)qp)\end{aligned}$$

Alternatively, if the expression was first simplified before applying Bernouilli:

$$\begin{aligned}\Pr[E] &= \frac{(1-p)^{(n-t)q} - (1-p)^{nq}}{1-(1-p)^{nq}} \\ &\approx \frac{(1-(n-t)qp) - (1-nqp)}{1-(1-nqp)} \\ &= \frac{tqp}{nqp} \\ &= \frac{t}{n}\end{aligned}$$

3.

$$\lim_{p \to 0} \frac{t}{n}(1-(n-t)qp) = \frac{t}{n}$$

We deduce that the approximation $\frac{t}{n}$ we used in our Monte Carlo simulation is "good enough" if the probability of a successful query is so small that multiple successful queries don't appear in the same round. Equivalently, the same effect is also observed if the network delivers messages with a very small delay, which makes multiple successful queries per round also unlikely.

References

Some helpful definitions are provided below. For the full definitions, consult the lecture notes and the bitcoin backbone paper.

Definition (The Proof-of-Work Inequality)**.**

$$H(B) \leq T$$

Definition (Chain Growth (formal))**.** An execution has *chain growth*, parametrized the growth interval s and the velocity τ if, for any rounds r_1, r_2 with $r_2 \geq r_1 + s$, and any honest party P, it holds that $|C^P_{r_2}| \geq |C^P_{r_1}| + s\tau$.

Definition (Common Prefix (formal))**.** An execution has *common prefix*, parametrized by $k \in \mathbb{N}$, if for all honest parties P_1, P_2 and for all times $r_1 \leq r_2$, it holds that $\mathcal{C}^{P_1}_{r_1}[:-k] \preceq \mathcal{C}^{P_2}_{r_2}$.

Definition (Chain Quality (formal))**.** An execution has *chain quality*, parametrized by the chunk ℓ and the quality μ if, for all honest parties P and round r, and for any positions $i < j$ in the chain with $j > i + \ell$, the ratio of honest to total blocks in $\mathcal{C}^P_r[i{:}j]$ is at least μ.

Definition (Honest Majority (formal))**.** An execution is said to satisfy *honest majority* with *honest advantage* δ if $t < (1-\delta)(n-t)$.

Definition (The Balancing Equation)**.** The balancing equation is given by

$$3\epsilon + 3f \leq \delta\,.$$

One possible configuration is $\epsilon = f = \frac{\delta}{6}$.

Definition (Typicality)**.** An execution is called *typical* if for all sets S of consecutive rounds, with $|S| \geq \lambda$, the random variables $X(S)$, $Y(S)$, $Z(S)$ are at most an ϵ error within their expectations:

- $(1-\epsilon)\mathbb{E}[X(S)] < X(S) < (1+\epsilon)\mathbb{E}[X(S)]$.
- $(1-\epsilon)\mathbb{E}[Y(S)] < Y(S) < (1+\epsilon)\mathbb{E}[Y(S)]$.
- $(1-\epsilon)\mathbb{E}[Z(S)] < Z(S) < (1+\epsilon)\mathbb{E}[Z(S)]$.

Definition (Chernoff Bound)**.** Consider a set of independent and identically distributed Bernoulli trials $\{X_i\}_{i\in[n]}$ with $\mathbb{E}[X_i] = p$. Consider their Binomial sum $X = \sum_{i=1}^{n} X_i$, and its expectation $\mu = \mathbb{E}[X] = np$. The probability of X deviating more than ϵ from $\mathbb{E}[X]$ is negligible in n. Concretely,

$$\Pr[X \leq (1-\epsilon)\mathbb{E}[X]] \leq e^{-\epsilon^2\mu/2}$$
$$\Pr[X \geq (1+\epsilon)\mathbb{E}[X]] \leq e^{-\epsilon^2\mu/3}\,.$$

Chain addressing notation.

- $\mathcal{C}^P_r$: The chain of party P at round r.

- $|\mathcal{C}|$: Chain length
- $\mathcal{C}[i]$: i^{th} block in the chain (0-based). The block height is i.
- $\mathcal{C}[-i]$: i^{th} block from the end.
- $\mathcal{C}[0]$: Genesis (by convention honest).
- $\mathcal{C}[-1]$: The tip.
- $\mathcal{C}[i{:}j]$: Chain chunk from block i (inclusive) to j (exclusive).
- $\mathcal{C}[{:}j]$: Chain chunk from the beginning and up to block j (exclusive).
- $\mathcal{C}[i{:}]$: Chain chunk from block i (inclusive) onwards.
- $\mathcal{C}[{:}-k]$: The stable chain.

Variables.

- κ: The security parameter, and size of the hash function output.
- H: The hash, modelled as a random oracle.
- n: The number of parties.
- t: The number of corrupt parties.
- δ: The honest advantage.
- q: Compute per party per round (number of allowed queries to the random oracle).
- T: The mining target.
- s: The chain growth interval, in units of time.
- τ: The chain velocity, in blocks per unit of time.
- k: The common prefix parameter, in blocks.
- ℓ: The chain chunk size required to ensure quality, in blocks.
- μ: The chain quality as a ratio.
- p: The probability of a successful query.
- f: The probability that a given round is successful.
- ϵ: The Chernoff error.
- λ: The Chernoff interval.
- X_r: The random variable indicating whether a round was successful.

- Y_r: The random variable indicating whether a round was a convergence opportunity.
- Z_r: The random variable counting adversarially successful queries in a round.
- $X(S)$: The random variable counting the number of successful rounds among rounds S.
- $Y(S)$: The random variable counting the number of convergence opportunities among rounds S.
- $Z(S)$: The random variable counting the number of adversarially successful queries among rounds S.