Principles of Blockchains
Princeton University,
Professor: Pramod Viswanath

Lecture 22

Theorical Questions

September 7, 2023

Contents

Chapter 1

Theorical Questions

1.1 Questions

1. The birthday paradox says that, counter-intuitively, the probability of a shared birthday exceeds 95% in a group of only 47 people. The birthday attack is a cryptographic attack that relies upon the birthday paradox to find collisions in the hash function. Now we are trying to find collisions in SHA-256. Please answer the following questions.
   - How many hashes do you need to calculate so as to guarantee a collision?
   - How many hashes do you need to calculate so that you can find a collision with probability 95%?
2. We create a new hash function called MySHA, which is just defned as SHA256 truncated to the frst 4 bytes (the frst 8 hex digits). Please fnd a hash collision in a MySHA. You should be able to produce a collision in no more than a few seconds.
3. Given a Merkel tree with 128 leaf nodes, how many nodes are there in the Merkel tree? What is the depth of the tree? What is the proof size? How about a Merkel tree with 200 leaf nodes?
4. When should a transaction be considered irreversibly confirmed in Bitcoin?
   - When all miners have received the transaction.
   - When it appears in a mined block.
   - When it appears in a mined block on the longest chain.
   - When it appears in a mined block, which has k blocks after it in the longest chain (for some value of k)?
5. Stochastic modeling of Bitcoin mining.
   - A very reasonable model for proof-of-work mining is the Poisson point process (cf., Chapter 3). Within this model, what is the distribution of the mining time between two consecutive blocks in Bitcoin? Write down the corresponding cumulative distribution function (CDF).
   - Using the data from this website, plot the empirical CDF of the inter-block mining time in the past two weeks and compare it with your model in first part.
6. In this question, we correct Nakamoto's table in the **Bitcoin whitepaper** (page 8). Considering the private attack discussed in this lecture, the goal of the adversary is to deconfirm a k-deep block. We assume the adversarial and honest mining processes are Poisson point processes with rates $\beta\lambda$ and $(1-\beta)\lambda$, respectively.
   - Simulate the private attack with your own choice of k, β and λ using Monte Carlo method. Does the success probability change with λ? why?
   - Simulate the private attack with the parameters in Nakamoto's table using Monte Carlo method. Do your simulation results match with values in Nakamoto's table?

- Nakamoto assumes that when the honest chain has length k, the length of the private chain will follow a Poisson distribution with expected value $\frac{k\beta}{1-\beta}$. Is this correct? verify your answer with Monte Carlo simulations.
- Derive a closed-form expression for the success probability by correcting the distribution in third part, and compare it with your simulation results in first part.

7. Difficulty-target conversion in Bitcoin. As of Dec. 28 2021, the mining difficulty of Bitcoin is $24,272,331,996,979.97$. In other words, only one in about this number of nonces that you try will work. How many leading zeros are there in the mining target (in hexadecimal)? Verify your answer by comparing it with the block hashes of that day.

8. In this lecture, we have seen that there are three core ideas to the Bitcoin difculty adjustment algorithm. In this question, we consider some other difficulty adjustment algorithms in Bitcoin and possible security vulnerabilities.

   - Consider a simple algorithm using only the heaviest chain rule, i.e., simply let the miners choose their own difficulty. Can you find attacks on this algorithm?
   - Consider an algorithm using the heaviest chain rule where difficulty is adjusted every 2016 blocks, but without any bound. Can you find attacks on this algorithm?
   - Consider the full algorithm used in Bitcoin. Argue why your attacks in first and second part do not work.

9. If it takes longer for blocks to be disseminated across the P2P network of Bitcoin, which of the following are true?

   - Forks become more likely.
   - Forks become less likely.
   - The chance of forks remains the same.

10. As the number of nodes in the P2P network of Bitcoin, N, grows, how does the time for a block to be disseminated to the network grow?

    - $O(1)$
    - $O(\log^N)$
    - $O(\sqrt{N})$
    - $O(N)$
    - $O(N^2)$

11. As the number of nodes in the P2P network of Bitcoin, N, grows, how does the total number of messages sent for a block to be disseminated to the network grow?

    - $O(1)$
    - $O(\log^N)$
    - $O(\sqrt{N})$
    - $O(N)$
    - $O(N^2)$

12. How to hash nonces: Assuming at each round of the mining process a random binary string x with the same length is given, and the goal is to find a binary nonce string y with the same length such that hash of a certain combination $\circ$ of x and y is smaller than a threshold τ, i.e., $H(x \circ y) \leqslant \tau$. For the following types of combinations, can you design attacks that do better than randomly guessing the nonce?

    - Bit-wise xor.
    - Bit-wise and.
    - Bit-wise or.

13. Fork and computation power: Assuming a static miner network where the computation power of each miner is fixed, a fork happens when two miners mine new blocks within a small interval $T\Delta$. Comparing miners with more computation power to miners with less computation power, are the probabilities of mining a fork different?

14. Let Δ be the maximum network delay in the Bitcoin network and τ be the expected time before a new block is mined. What should be true to guarantee safety?

    - $\Delta \ll \tau$
    - $\Delta < \tau$
    - $\Delta \approx \tau$
    - $\Delta > \tau$
    - $\Delta \gg \tau$

15.

1.2 Answers

1. - To guarantee a collision in SHA-256 hashes, you need to calculate 2^{128} hashes.
     SHA-256 outputs a 256-bit hash. For there to be a collision (two inputs hashing to the same output), the hash space needs to be fully explored. SHA-256's hash space is 2^{256} possibilities. So to guarantee a collision, we need to generate 2^{128} unique hashes.
   - To have a 95% probability of finding a collision in SHA256 hashes, you need to calculate approximately 2^{64} hashes.
     This relies on the birthday paradox. The birthday paradox says that the probability of a collision in a set of randomly generated values exceeds 50% when the number of values exceeds the square root of the size of the problem space.
     For SHA-256, the problem space is 2^{256}. Taking the square root of 2^{256} is approximately 2^{128}. With 2^{64} hashes, we exceed the square root threshold and obtain around a 95% probability of a collision occurring.

2. Answer is written in Q2.py.

3. - For a Merkel tree with 128 leaf nodes:
     - Number of nodes in the tree: $2 * 128 - 1 = 255$
     - Depth of the tree: $\log_2^{128} = 7$
     - Proof size: 7 (the number of hashes needed to validate a leaf)
   - For a Merkel tree with 200 leaf nodes:
     - Number of nodes in the tree: $2 * 200 - 1 = 399$
     - Depth of the tree: $log_2^{200} = 8$
     - Proof size: 8 (the number of hashes needed to validate a leaf)

   Some explanations:
   A Merkel tree with N leaf nodes will have $2N - 1$ total nodes. This is because each non-leaf node has two child nodes, resulting in a complete binary tree.
   The depth of a Merkel tree is the number of branches from the root to the deepest leaf node. This can be calculated as log_2^N where N is the number of leaf nodes.
   The proof size is equal to the depth, as it represents the number of hashes needed to validate a leaf when traversing the tree from the root.

4. The correct answer is when it appears in a mined block, which has k blocks after it in the longest chain (for some value of k).
   In Bitcoin, a transaction is not considered fully confirmed until it is buried under several blocks in the longest blockchain. The reasoning is: Requiring k additional blocks on top provides a high degree of certainty that the transaction will not be reversed. As k increases, the probability of an orphaned block or chain reorg

decreases exponentially.
The standard for irreversible in Bitcoin is when a transaction has 6 confirmations, meaning it is buried under 6 additional blocks. But a higher k value results in an even higher degree of certainty.
So in summary, only requiring the transaction to be deep in the longest chain, not just received or in any single block, makes it irreversible on the Bitcoin network.

5. - Within the Poisson point process model of Bitcoin mining:

     - The distribution of the time between consecutive blocks is exponential.
     - The cumulative distribution function (CDF) is:

       $$F(t) = 1 - e^{-t\lambda}$$

       Where λ is the expected rate of block discovery, which in Bitcoin is approximately equal to the reciprocal of the intended average block time (10 minutes or 600 seconds).
   - Not answered yet :))

6. - Simulation is written is Q6-a.py. The success probability decreases with λ because higher mining power (λ) makes it harder for the attacker to outpace the honest chain.
   - Not answered yet :))
   - No, Nakamoto's assumption that the private chain length follows a Poisson distribution is not correct. The correct distribution is a binomial, as it is the result of repeating Bernoulli trials (mining a block or not).
   - With the correct binomial distribution, we can derive the closed form expression for success probability by calculating the CDF of the binomial and setting it equal to $\frac{1}{k}$.

7. Not answered yet :))

8. - In a simple algorithm where miners choose their own difficulty without any regulation or coordination, there are several potential attacks:
     (a) **Selfish Mining Attack:** Miners might strategically lower their difficulty to mine blocks more quickly and increase their own rewards. They can then release these blocks to the network in a way that disrupts the natural difficulty adjustment process, leading to inconsistent block generation intervals and potential forks.
     (b) **DoS Attacks:** Miners might set extremely high difficulties, effectively refusing to mine. This could disrupt the normal operation of the network by slowing down block generation to a crawl.
     (c) **Economic Attacks:** Miners can engage in strategic behavior by adjusting their difficulty to influence the mining rewards and transaction fees they receive, potentially leading to imbalanced incentives and security issues.
   - In an algorithm where difficulty is adjusted every 2016 blocks without any bound or constraints, the main problem is that there's no safeguard against rapid changes in difficulty. This could lead to two significant issues:
     (a) **Wild Fluctuations:** Without any bounds on difficulty adjustments, miners could quickly change the difficulty based on their interests. This might lead to rapid oscillations in block generation times, making the network unreliable and potentially causing significant forks.
     (b) **Security Vulnerabilities:** Miners could strategically manipulate difficulty adjustments to weaken the security of the network. For example, they might lower the difficulty to encourage more miners to join, only to raise it dramatically afterward, making it extremely difficult for the network to keep up.
   - The full Bitcoin difficulty adjustment algorithm, as designed by Nakamoto, incorporates several important mechanisms to address these issues:
     (a) **Regular Adjustment Intervals:** Difficulty is adjusted every 2016 blocks (approximately every two weeks). This ensures that the network can adapt to changes in computational power without experiencing wild fluctuations.

(b) **Bound on Adjustment:** The algorithm includes a mechanism to limit the maximum adjustment to a factor of 4 in either direction. This prevents rapid, extreme changes in difficulty.

(c) **Proof of Work (PoW):** The security of the Bitcoin network is maintained through PoW, which provides incentives for miners to behave honestly. The difficulty adjustment algorithm doesn't change the fundamental security properties of PoW.

(d) **Longest Chain Rule:** The heaviest chain rule is still a fundamental part of Bitcoin's consensus mechanism. However, the difficulty adjustment algorithm ensures that the heaviest chain is based on a predictable and stable block generation rate.

9. The answer is forks become more likely.
When it takes longer for blocks to be disseminated across the P2P network of Bitcoin, the chance of multiple miners solving proof-of-work puzzles simultaneously and broadcasting their blocks at roughly the same time increases. This increases the likelihood of temporary forks in the blockchain. As more miners receive and build upon different blocks, it becomes harder for the network to reach a consensus quickly, potentially leading to forks.

10. The answer is $O(\log^N)$
As the number of nodes in the P2P network of Bitcoin, N, grows, the time for a block to be disseminated to the network generally grows logarithmically. This is because with more nodes, there are more potential pathways for information to travel, but the growth in time is typically not linear ($O(N)$) because of the efficient P2P network topology and communication mechanisms in place. It's closer to logarithmic growth due to the network's structure and the way information spreads through it.

11. The answer is $O(N)$
As the number of nodes in the P2P network of Bitcoin, N, grows, the total number of messages sent for a block to be disseminated to the network typically grows linearly ($O(N)$). This is because each node needs to send the block to every other node in the network, resulting in a linear relationship between the number of nodes and the number of messages exchanged.

12. For each of the specified types of combinations, here are some considerations on whether attacks can do better than randomly guessing the nonce:

- **Bit-wise XOR:**
  - In this case, the goal is to find a nonce y such that $H(xXORy) \leqslant \tau$.
  - XOR is a bitwise operation that returns 1 if the bits being compared are different and 0 if they are the same.
  - Since the hash function is typically designed to produce a uniformly distributed output for small changes in the input (avalanche effect), trying to predict how XORing a specific nonce with x will affect the hash output is very difficult.
  - Therefore, for the bit-wise XOR combination, it is unlikely that any attack can consistently perform better than random guessing the nonce.
- **Bit-wise AND:**
  - In this case, the goal is to find a nonce y such that $H(xANDy) \leqslant \tau$.
  - Bit-wise AND returns 1 if both corresponding bits in x and y are 1, otherwise it returns 0.
  - An attack in this scenario could potentially perform better than random guessing if there are patterns in the hash function's output that can be exploited. For example, if the hash function has specific weaknesses related to the AND operation, an attacker might find a nonce that, when ANDed with x, produces predictable bits in the hash output.
  - However, if the hash function is designed properly and does not exhibit such patterns, then it would be challenging for an attack to do better than random guessing.
- **Bit-wise OR:**
  - In this case, the goal is to find a nonce y such that $H(xORy) \leqslant \tau$.
  - Bit-wise OR returns 1 if at least one of the corresponding bits in x and y is 1.
  - Similar to bit-wise AND, whether an attack can do better than random guessing depends on the specific properties of the hash function.

  - If the hash function has known weaknesses related to the OR operation, an attacker might exploit those weaknesses to find a nonce that produces the desired hash output.
  - However, if the hash function is robust and well-designed, it should not exhibit predictable patterns when performing bit-wise OR, making it difficult for an attack to do better than random guessing.

13. In a static miner network where the computational power of each miner is fixed and known, the probability of mining a fork is not inherently different between miners with more computational power and miners with less computational power. This is because, in a fair and deterministic mining process, each miner's probability of mining a block (or a fork) is proportional to their computational power relative to the total computational power of the network.
    Here's how it works:

    - **Mining Probability:** In a Proof-of-Work (PoW) blockchain like Bitcoin, miners compete to solve a cryptographic puzzle (finding a nonce) to add a new block to the blockchain. The probability of a miner successfully solving this puzzle and mining a valid block is directly proportional to their computational power. Miners with more computational power have a higher chance of solving the puzzle faster.
    - **Network Consensus:** To maintain consensus in the blockchain network, miners must agree on the "longest chain" rule. The longest chain is the one with the most accumulated computational work, not necessarily the one with the most blocks.
    - **Fork Occurrence:** Forks occur when multiple miners successfully mine new blocks within a short time frame ($T\Delta$). This can happen when miners solve the puzzle simultaneously or nearly simultaneously. It's important to note that forks are typically resolved by the network through a consensus mechanism.

14. Not answered yet :))