Cryptographic Basics

Cryptographic Hash Functions and Merkle Trees

Alice and Bob want to play rock-paper-scissors over a peer-to-peer connection. To prevent cheating, they want to use their knowledge of cryptography to devise a commitment scheme based on hashing. To start out, they consider possible hash functions.

1. At one point, Bob proposes the following function:

$$h(x) = x + 17$$

Explain why this function is not a hash function.

2. Next, they fix Bob's mistake and consider the following simple hash function:

$$g(x) = (x + 17)\ mod\ 1024$$

Still, they deem it unsuitable. Recall the key properties of cryptographic hash functions from the lecture. Name the property this function violates and briefly explain why.

Given some time, Alice and Bob come up with some arbitrary cryptographic hash function h and the following scheme. One round looks like this:

(a) Both secretly choose one option: rock, paper, or scissors.
(b) Both compute the hash $h_i = h(choice_i)$ and send it to each other.
(c) When they reveal their choice to the other, the other can verify that the commitment was made before the reveal by hashing the revealed choice and comparing to the previously received hash.

3. Where is the flaw in this scheme?

4. Propose a way to fix this scheme.

Since Alice and Bob are busy students, they decide to save time and expand their scheme to support playing multiple games per round of their scheme. Naively sending n commitments at once leads to a linear increase in required hashes and thus an increase in network traffic. Well versed in cryptography, they want to decrease the required network traffic by using Merkle trees.

5. Given the use of Merkle trees, what is the minimum number of hashes Alice has to send to Bob to **commit** to rock, paper, scissors for a round consisting of 16 games.

6. Alice and Bob agree to play a round consisting of 3 games. Draw the Merkle tree over Alice's three hashes $h(c_1) = 1, h(c_2) = 3, and\ h(c_3) = 7$. For this construction assume:

$$h(x) = x\ mod\ 8$$

and use addition to combine hashes (instead of concatenation).

Search Puzzle

We want to design a search puzzle using the puzzleID “BBSE_E01” and the SHA_256 hash function. Assume that the target difficulty $d = 2^{240}$ (i.e., the accepted solution space is defined in $[0, 2^{240} - 1]$).

1. What is the probability of finding a correct input on the first try? Given that a computer can generate 2^{15} hashes per second, how many seconds should elapse before the computer can be expected to find a correct solution?
   Hint: Think about Bernoulli Trials and Geometric Distribution.

2. What is the value x that solves the puzzle? How long does your computer execute until it finds a result? Select your favorite programming language and develop this search puzzle. If you do not have a preference, you can use JavaScript or TypeScript, as we will use them in the practical Ethereum exercises.
   Hint: The last accepted solution ($d - 1$) has four leading zeros in hex representation (“0000fff...”).

3. Three computers (hashing power $A = 50\%, B = 30\%, C = 20\%$, overall 100,000 hashes per second) participate in this search puzzle. All computers use the same strategy to solve the puzzle: increment x with $x + 1$. Which one wins the search puzzle?

4. Suggest possible ways for the losing computers to change their own strategy in order to increase their chances of winning.

5. In this part, assume that the computers did not change their strategies. How can the puzzle be changed so participants can win according to their hash power?