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.

Deterministic: Given same input, it should produce the same output.

Premage Resistance: It should be computational infesiable to find an input that maps to a given hash output.

Second preimage resistance: Given an input, it should be computationally infesible to find another input that produce the same hash output.

Collision Resistance: ---- infesible to find two distinct values to produce the same hash output

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

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

Explain why this function is not a hash function.

Lack of unpredictability: A good hash function should produce an output that is significantly different even for small changes on input. However, "$h(x) = x+17$" output is easly predictable because it's linear function. Therefore, attacker can exploit funct easly.
Easy Reversibility: It should be computationally infesible to retrive the original input given hash output (preimage resistance)

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.

Weak collision Resistance: It mapping same output with different inputs. A secure hash function should make it computationally infesible to find two distinct inputs that produce same hash output.
Preimage Resistance: Still attacker can find input by given input x by calculating $x = (y-17) \bmod 1024$

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?

. Alice or Bob could potentially precompute the hash values for all choices and cheat by picking a winning move after seeing the other hash commitments.

4. Propose a way to fix this scheme.

To fix this Alice and Bob can add a random value (nonce) to their choices when hashing. This makes it harder for an opponent to deduce the choice from the hash commitment and prevents cheating.

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.

A balanced binary Merkle tree with 16 leaf nodes has four levels. (including the leaf node)

Level 1: 16 internal nodes
Level 2: 8 internal nodes
Level 3: 4 internal nodes
Level 4: 2 internal nodes
Merkle root: 1 root node

to commit the 16 games, Alice only needs to send 1 hash (the Merkle root. However, during the verification process, Alice will need to send 4 sibling hashes for each game in order to proove. So $4 * 16 = 64$ hashes in total.

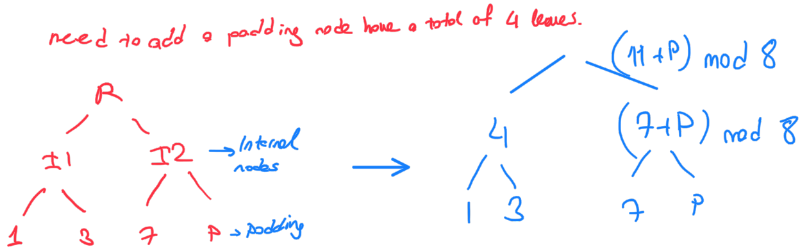
6. Alice and Bob agree to play a round consisting of 3 games. Draw the Merkle tree over Alice's three hashes $h(c_1) = 1, h(c_2) = 3, and\ h(c_3) = 7$. For this construction assume:

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

$c_1 = 9, 17$
$c_2 = 11, 19$
$c_3 = 15, 23$

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

Since we can't create perfectly balanced three with 3 leaf, we'll need to add a padding node have a total of 4 leaves.



Proof of work (PoW): It is a consensus (fikir birliği) algorithm used in blockchain network and other distributed systems to determine malicious behavior and maintain network security.

* It requires miners to solve computationally intensive puzzles in order to validate transactions and (create new block ⇒ refers to process of adding new sets of data (usually transactions) to the blockchain.)

Blocks contains a collection of transactions, a reference to previous block, a timestamp and other relevant info.

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.

   SHA generates 256-bit hash output, which means there are 2^{256} possible hash values. Since difficulty is 2^{240}, there are 2^{240} acceptable hash values.

   $P(\text{Success}) = 2^{240} / 2^{256} = 1/2^{16}$

   Expected Time to find correct solution: Geometric distribution
   $E(\text{trials}) = 1/P(\text{Success}) = 2^{16}$

   $E(\text{time}) = E(\text{trials}) / \text{hashing rate} = 2^{16}/2^{15} = 2$ second.

   computer can be expected to find correct solution in 2 second.

Target Difficulty: It is predefined threshold used in PoW blockchain consensus algorithm to determine the level of computational effort to solve cryptographic puzzle and add new block to blockchain.

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?

Although A has the highest Hash power, it is not guaranteed that A can be always winner. Because process is undeterministic.

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

Some suggession for losing computers to increase their chances of winning.

- Non-overlapping search space: Divide the search space among computers to avoid searching same value.
- Randomized Search: choose random values of X instead of sequential increments.
- Parallel Search: Utilize multiple cores or threads for faster search.
- Collabration: Computers B and C could pool resources and work together.
- Optimization: Improve hashing algorithm, hardware or system performance

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?

Divide the search time into discrete time intervals (e.g., 1-second intervals).
Assign each participant a portion of the interval proportional to their hashing power. For example, in a 1-second interval, computer A (50% hashing power) gets 0.5 seconds, computer B (30% hashing power) gets 0.3 seconds, and computer C (20% hashing power) gets 0.2 seconds.
During each participant's assigned time, they search for the solution, incrementing x sequentially or using another strategy.
When a participant's time expires, they pause their search and wait for their next assigned time interval.
Repeat this process until a participant finds a valid solution.