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.

The function violates compression.

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.

Pre-image resistance is violated.
It is computationally feasible to find an input x given output g(x) s.t. *x = g(x) - 17.*

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?

There are only 3 choices that are hashed without salt allowing easy brute-force attacks.

4. Propose a way to fix this scheme.

Concatenate the choice with a random number before hashing.

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.

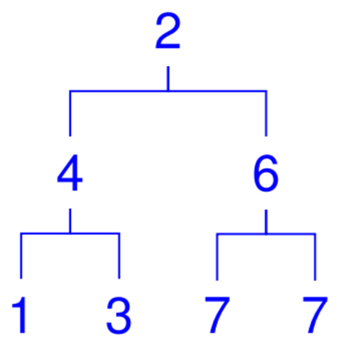
5. Given the use of Merkle trees, what is the minimum number of hashes Alice has to send to Bob to **commit** to rock, paper, scissors for a round consisting of 16 games.

By sending the root of the Merkle tree to Bob, Alice commits to her 16 answers included in the leaves of the Merkle tree. Hence, the minimum number is 1 (Independent of n)

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

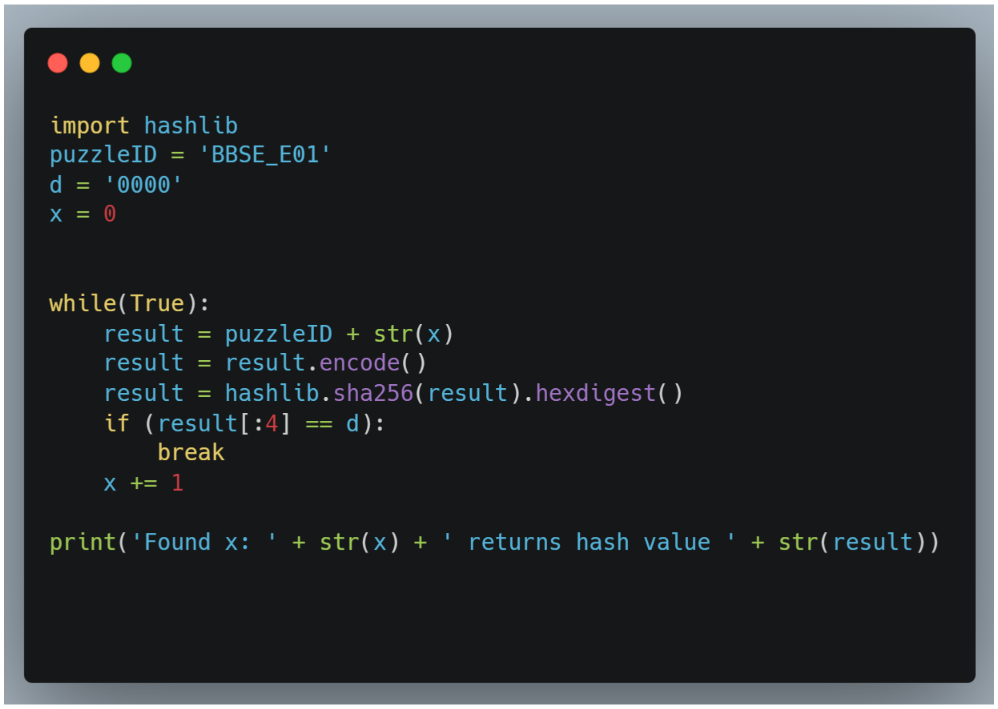
We want to design a search puzzle using the puzzleID "BBSE_E01" and the SHA_256 hash function. Assume that the target difficulty $d = 2^{240}$ (i.e., the accepted solution space is defined in $[0, 2^{240} - 1]$).

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.

   (a) Since the solution space contains 2^{240} correct inputs (including 0), the probability of finding a correct input in the first try is $\frac{2^{240}}{2^{256}} = 2^{-16}$.

   (b) If we treat attempts to solve the puzzle as Bernoulli trials (i.e., series of independent success/failure experiments with fixed probabilities), then, Geometric distribution tells us that the expected number of trials needed before finding the first correct input is $1/p$ where p is the probability of success. Hence, we need 2^{16} tries. As the computer can only generate 2^{15} hashes per second, it will take $\frac{2^{16}}{2^{15}} = 2^1$ seconds to find a solution.

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…").



```python
import hashlib
puzzleID = 'BBSE_E01'
d = '0000'
x = 0


while(True):
    result = puzzleID + str(x)
    result = result.encode()
    result = hashlib.sha256(result).hexdigest()
    if (result[:4] == d):
        break
    x += 1

print('Found x: ' + str(x) + ' returns hash value ' + str(result))
```

$7852 \rightarrow sha_{256}(\text{BBSE\_E017852}) = 000072d930284346c3d54c5cda0306900078c9be695b24d89$
$9d5ebeb2057ebcf$
Further information about the program: The used language is Python. We import the built-in hashlib module to be able to use the SHA256 hash function. In the while loop: The concatenation in the first line of the while loop works in Python with a plus (+) instead of our known notation (||). In the second line of the loop we convert the string into bytes using the encode() method (it uses UTF-8 encoding by default) in order to be accepted by the hash function. Then we use the hexdigest() method to get the hash result in hexadecimal. In this simplified program, we do not compare the numbers (difficulty and hash), but instead, we check if the hash starts with a certain amount of zeros. If that is the case, the while loop exits and displays the found number.

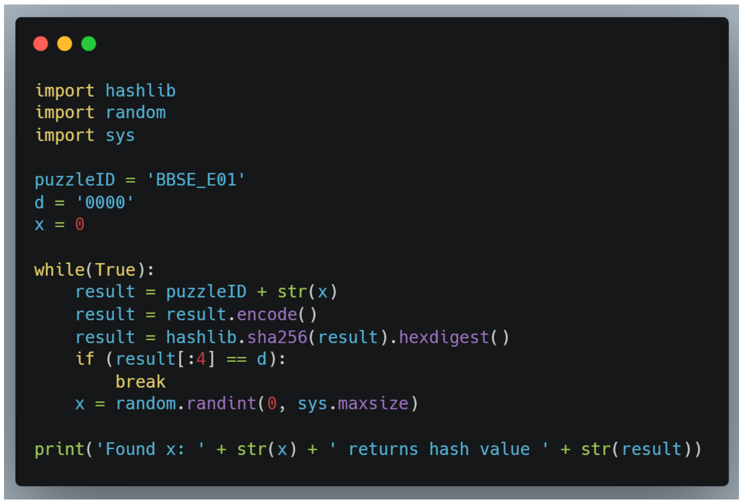
3. Three computers (hashing power $A = 50\%, B = 30\%, C = 20\%$, overall 100,000 hashes per second) participate in this search puzzle. All computers use the same strategy to solve the puzzle: increment x with $x + 1$. Which one wins the search puzzle?

The computer A wins, as he will reach x=7852 as the first computer. It is the fastest.

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

They need to change their strategy for iterating over the puzzle. Possible ways are:

- Do not iterate with x+1, but start with another number for x.
- Do assign x a random value every time.



```python
import hashlib
import random
import sys

puzzleID = 'BBSE_E01'
d = '0000'
x = 0

while(True):
    result = puzzleID + str(x)
    result = result.encode()
    result = hashlib.sha256(result).hexdigest()
    if (result[:4] == d):
        break
    x = random.randint(0, sys.maxsize)

print('Found x: ' + str(x) + ' returns hash value ' + str(result))
```

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?

   We can introduce a variable in the $PuzzleID$ that is user-dependent. For example, the hashing process could look like this: $Hash(PuzzleID||computerName||x)$. This way, each participant will solve their own puzzle and have a chance to win, according to their hashing power. When they try to solve the same puzzle using the same strategy, the one with the highest hash power will always reach the solution first.