Information Theory Lecture 2019

GIST

Heung-No Lee

September 9th 2019

Application of Probability Theory to Bitcoin Crypto Puzzles

New problems on crypto puzzles were introduced in class. The proof-of-work (PoW) in the Bitcoin blockchain uses a crypto puzzle. It is interesting to define some random variables, evaluate the probability of PoW success, and see how difficult the puzzle can be made.

A Hash Algorithm (HA) maps an input file to a hash. A hash is an integer. The hash output of the HA is a summary of the input file. Any integer smaller than a certain fixed maximum value can be expressed as a fixed length binary string. For example, the integer 4 and the binary string 111 are equivalent. All binary strings of length 256 are integers smaller than 2^{256}. HA-256 is one function in a HA family.

Definition. (HA-256) Let us refer to $F(\,)$ as a HA-256 function, i.e.,

$$F : \mathcal{X} \to \mathcal{Y}. \tag{1.1}$$

$\mathcal{X}$ is the set of text files. $\mathcal{Y}$ is the set of binary strings of length 256. This function is a *oneway* function such that going *forward* is easy but going backward, calculating the *inverse* of an output, is very difficult. Given an output, unlike the usual functions $ax+b$ or e^x we have seen in grade schools, one cannot find out what the corresponding input was. With a tiny difference in the input, the output changes drastically. This better be understood by some examples given next.

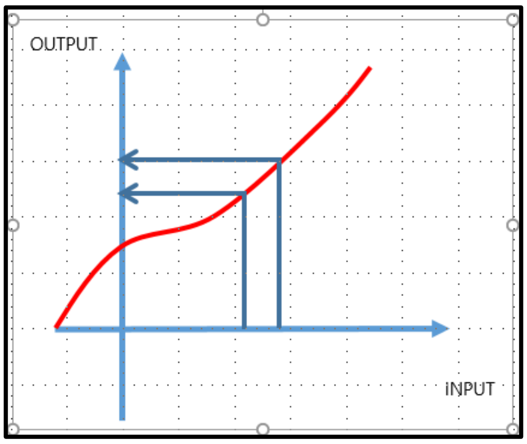


Figure 2: A usual function.

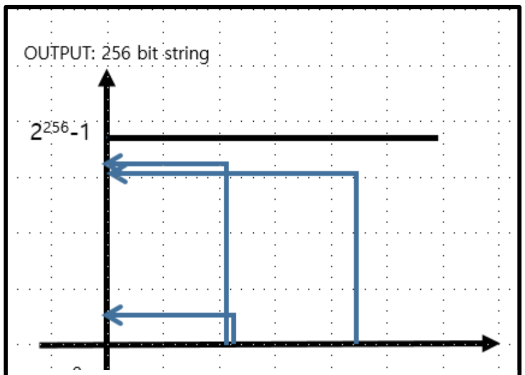


Figure 1: Input and output relation of a Secure Hash Function. It takes a file as input and outputs a 256 bit string. The illustration is for any input file whose size is up to 1Megabyte.

This type of functions is useful in cryptography. When you log into a portal site server, you type in your passwd along with your user name. Your typed-in passwd is never supposed to be saved at the server. Only its hash is allowed to be saved. The typed-in passwd shall be translated into its hash which is compared to the hash saved at the server under your user name.

One example of HA is SHA-256 of NIST. See https://www.nist.gov/publications/secure-hash-standard.

Here is the demo of SHA-256. https://www.xorbin.com/tools/sha256-hash-calculator.

Example:

INPUT: Information Theory 2019 GIST Heung-No Lee

Hash:
0723d1146eea5305d1b4467709519b55a3dd38d0afdbbe60acf168db873f3da5

Example:

INPUT: Information Theory 2019, GIST Heung-No Lee

Hash:
3a91ecc749748e2f120e2a02e2d331e7ce90a9ba254d6f0fbbd3ef69c97297e9

Some properties of 256 SHA functions

1. With the same input, it always gives the same output.
2. Say you had a first input and output. You give a second input which is a tiny difference from the first input, it gives a random looking output with a large difference to the first output.
3. As a result, given an output, the input cannot be correctly obtained.
4. It is a many to 1 function. The output space is usually much smaller than the input space.
5. But the output space is large enough, $2^{256} \approx 10^{77}$, so that it is almost impossible to find a collision case.
6. It uses many rounds of additions and multiplications with many (huge number) modulo operations.
   A. Ex) Let us consider $F(x) = 7x + 6 \bmod 5$. Let $x = 5$; $y = 7 \times 5 + 6 = 41 \bmod 5 = 1$.

Let $x = 5$; $y = 7 \times 4 + 6 = 34 \bmod 5 = 4$. Now imagine doing this at a large modulo operation and a large number space. Refer to NIST web-site for further details.

Definition. (Proof-of-Work and its Success) Given a target number $2^1 \leq \texttt{Target} \leq 2^{256}$, Proof-of-Work(PoW) is defined precisely. The *working* part is to finding an input *x* satisfying the following inequality:

$$F(x) < \texttt{Target} \tag{1.2}$$

and the proof part is to keep the record of found *x* as a proof-of-work done.

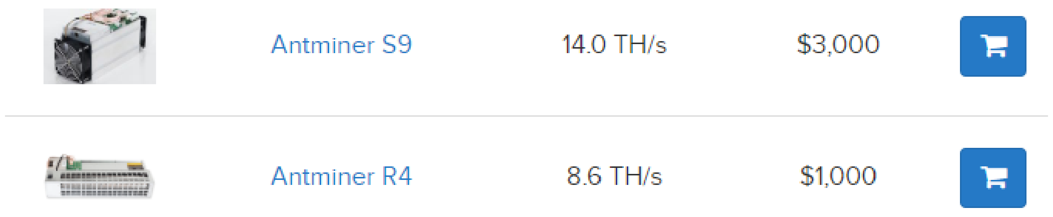
It shall be noted that given *x*, one can easily verify if its hash satisfies (1.2) or not. If it does not satisfy, repeat the work until a good input *x* is found. Thus, a single round of PoW puzzle is completed eventually with a single good input *x* and its hash are found and recorded. Then, the question is to ask when that first success occurs. If it is not found at the first trial, input is changed and tried again until we get the first PoW success. With a single success, a single round of PoW puzzle is completed. This process can be repeated for the next round of PoW puzzle.

Example) Proof of Work can be used to prevent spammers from sending you spam e-mails.

- Suppose you set your mail box to present you only those e-mails with PoW attached.
- For example, a protocol is enforced such that the sender has to provide a hash proof.
- The hash is the output of the hash function which takes both the content of his e-mail and a counter. The counter can be used to vary the output hash while his e-mail content unchanged. The sender has to do the work to find a right hash. To find a right hash, the sender has to find a right counter.
- Now you place a threshold condition to the hash output. A hash proof is found had the spammer found the counter which results in a good hash. Good hash here means a hash smaller than a certain threshold.
- For example, the threshold can be set to 2^{255}. Then, the spammer has to run his cpu to find a right counter. How many cycles would it take on the average? Two cycles. Repeat the question with the threshold set at 2^{253}. Eight cycles.
- The recipient, you want to receive only those e-mails with a good hash proof attached. The recipient just needs to check the proof in a single hash cycle. You can check the hash proof and the e-mail content and see if they satisfy the threshold requirement. Thus, the recipient would not have to spend too much time.

This Proof-of-Work system is used in Bitcoin network. Bitcoin is a peer-to-peer network of computers. Millions of peer nodes are competing to make a block. A block contains Bitcoin transactions. PoW is used to have the peers compete to make a block. A block is generated in every 10 minutes. The first peer node which finds a good hash in a block generate cycle is to generate a certain number of new bitcoins for the block. This is done to make the network to be decentralized (one cpu-one vote).

Definition. (Mining Chip and Hash Rate) A mining chip is designed specifically to solve the PoW puzzle as fast as possible. Its capability is expressed in terms of the hash rate [hash/sec]. The hash rate of a mining chip is to indicate how many input-output cycles of the HA, $F(x) = y$, the chip can process within a unit time.



Antminer S9	14.0 TH/s	$3,000	
Antminer R4	8.6 TH/s	$1,000	

Figure 3. The hash rate of Antminer S9 is 14 Tera hash/sec.

https://www.buybitcoinworldwide.com/mining/hardware/

Random Experiment.

We can model the PoW puzzle as a repeated random experiment.

Since F is a *oneway* function, the working part is tedious and difficult. The only way to find a satisfying answer x is by repeating the random experiment.

A single random experiment is to choose an input x from $\mathcal{X}$, obtain an hash $y = F(x)$ of x. The outcome of this experiment is y. Choosing an input file is carrying out this experiment. Given the output, we then see if the output belongs to a certain subset or not. This is a single trial (hash cycle) of experiment. In this PoW puzzle, it is to see if this output y satisfies the inequality or not. If it satisfies, we have success; otherwise, failure. Hash cycle is continued until we get the first success.

Definition. Bernoulli trial with success probability p**.** Let `Target` be given, i.e., $2^1 \le \texttt{Target} \le 2^{256}$. With a single selection of a file $x \in \mathcal{X}$, we obtain $F(x) = y$. The outcome $y \in \mathcal{Y}$ belongs to either the event of *success*, i.e., $\{y : y < \texttt{Target}\}$ or that of *failure*, i.e., $y \in \{y : y \ge \texttt{Target}\}$.

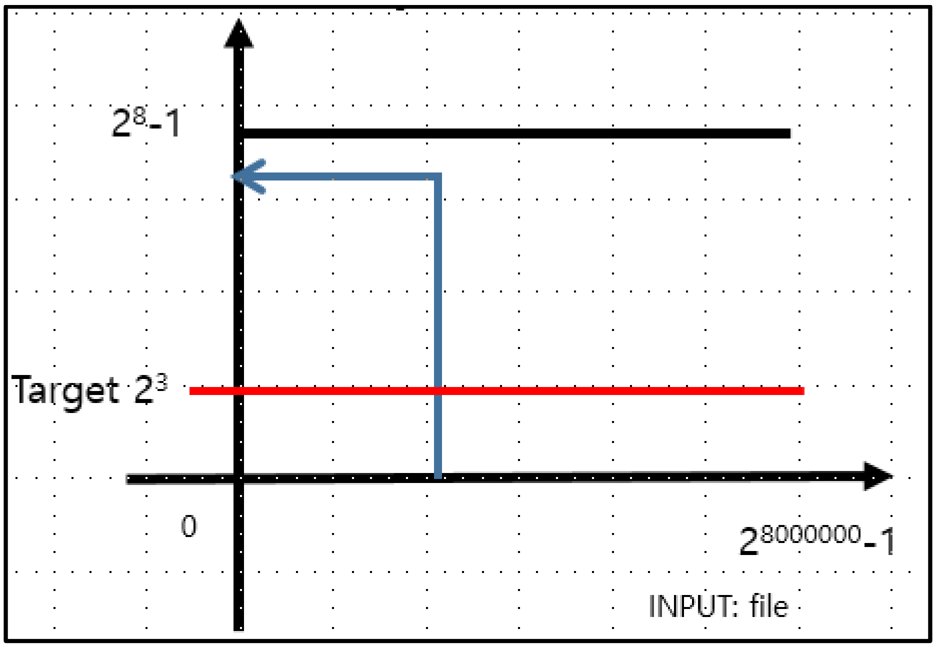


Figure 4: Suppose the hash function outputs a hash of 8 bits in this example. Target in the picture is the threshold. The set is divided into two mutually exclusive sets, a good set and a bad set.

Given the characteristic of SHA, the probability p of success in a single trial can be given by

$$\begin{aligned} p &= \frac{\left|\{y : y < \mathtt{Target}\}\right|}{|\mathcal{Y}|} \\ &= \frac{\mathtt{Target}}{2^{256}}. \end{aligned} \tag{1.3}$$

We now can define a Bernoulli random variable. This random experiment is to be repeated in a series of repetitive trials. Thus, let us define B_n the n-th Bernoulli random variable.

Definition. A series of Bernoulli random variables B_n for each trial index $n = 1, 2, 3, \cdots$ can be defined. The n-th Bernoulli trial is compactly described by the Bernoulli random variable $B_n : \mathcal{X} \to \{1, 0\}$ with parameter p. Use 1 for the event of success and 0 for that of failure. That is, $\Pr\{B_n = 1\} = p$ and $\Pr\{B_n = 0\} = 1 - p$. We may assume under all practical relevance that they are i.i.d. for PoW puzzles such that the random variables are independent with each other and the same parameter p for each trial can be used.

We are now interested in calculating the probability of solving the PoW puzzle for the first time at the k-th trial. For this, it is convenient to design a new random variable K.

Lemma. (RV K is to denote the first PoW success at the k-th trial) Let K be the trial index at which the first PoW success occurs. Then, for $k = 1, 2, 3, \cdots$, K is random variable with the geometric distribution:

$$\Pr\{K = k\} = (1 - p)^{k-1} p\,. \tag{1.4}$$

Its cumulative distribution function is given by

$$\begin{aligned}\Pr\{K \le k\} &= p + (1-p)p + (1-p)^2 p + \cdots + (1-p)^{k-1} p \\ &= \sum_{j=1}^{k} (1-p)^{j-1} p \\ &= 1 - (1-p)^k\end{aligned}\,. \tag{1.5}$$

The last step can be proven.

The mean and the variance are given as

$$\mathbb{E}\{K\} = \frac{1}{p} \quad \text{[hashes/success]} \tag{1.6}$$

and

$$\operatorname{Var}(K) = \frac{1-p}{p^2}\,. \tag{1.7}$$

Q.E.D.

We note that the hash period – the duration between giving an input to HA and getting an output $F(y)$ – is short in time. For example, it is 1 pico sec/hash for an Antminer S9. A single hash is obtained in a time interval on the order of 1 trillionth of a second, $O(10^{-12})$ sec. The hash rate of mining chip is on the order of 1 Tera hash/sec. Within this extremely short time duration $O(10^{-12})$ sec, an Antminer S9 obtains the hash of an input.

Now, we would like to embed the concept of time into the random variable of PoW success. The random variable K sort of already gives us the concept of time in the sense of number of hash cycles. But it only gives us the trial index k at which the first successful hash is produced. So, we want to improve it by embedding the time duration.

Let T_c be the hash time [sec/hash]. It is the inverse of hash rate of a mining chip.

Theorem. (PoW Success Time is Exponential Random Variable) Let $S = KT_c$ be the random variable to denote the PoW Success Time. To derive the distribution, we use calculus takings the limits $T_c \downarrow 0$ and $p \downarrow 0$ and obtain the following result:

$$\begin{aligned}\Pr\{S>t\} &= \Pr\left\{K>k\right\} \\ &= \left(1-p\right)^{k} \\ &= \left(1-p\right)^{\frac{1}{p}\frac{p}{T_c}kT_c} \\ &= \left(1-p\right)^{\frac{1}{p}\left(\lambda kT_c\right)} \\ &= e^{-\lambda t}\Big|_{t=kT_c}\end{aligned} \tag{1.8}$$

where we define $\lambda := \frac{T_c^{-1}}{p^{-1}}$. With $p \downarrow 0$, we note $(1-p)^{\frac{1}{p}} = \mathrm{e}^{-1}$. With small T_c, we can treat $t = kT_c$ to be a continuous time. *S* is exponentially distributed random variable. Q.E.D.

Definition. (Average Success Rate) The parameter λ is the average number of PoW successes $\lambda := \frac{T_c^{-1}}{p^{-1}}$ in a unit time. The unit of lambda is $\frac{\text{[hashes/sec]}}{\text{[hashes/success]}} = \text{[success/sec]}$.

We note that $1/\lambda$ is the average time-interval of a single PoW success.

We have defined the average success rate for a chip. This can also be made for a P2P computer network as well.

Homework

1. Find the hash value of your name and record it in your book. Place a dot at the end of your name, get the hash again. See how the two hashes are different.
2. Let $K_1 := K - 1$. Define this new variable. Obtain the distribution, mean and variance.
3. Explain why we may assume that B_n s, $n = 1, 2, 3, \cdots$, are i.i.d. for PoW puzzles.
4. Prove the last step in (1.5).
5. The PoW success trial is repeated until the first success occurs. Let K denote the number of trials required. Use $\texttt{Target} = 2^{250}$. One may find the following items useful: $\sum_{n=0}^{\infty} r^n = \frac{1}{1-r}$, $\sum_{n=0}^{\infty} nr^n = \frac{r}{(1-r)^2}$.

   (a). Provide the distribution of K.

   (b). Find the *entropy* of K in bits.

   (c). Define the success event for using $\texttt{Target} = 2^{250}$.

   (d). The random variable K is drawn according to this distribution. Find an "efficient" sequence of yes-no questions of the form, "Does the outcome of K belong to a set?"

   (e). Compare the expected number of questions required to determine *K*.

6. Let k_{out} be a large natural number and define an outage set, $\mathcal{K}_{out} := \{k \in \mathcal{N} : k > k_{out}\}$. Let Y be a random variable:

$$Y = \begin{cases} 1 & K \le k_{out} \\ 0 & \text{o.w.} \end{cases} \tag{1.9}$$

   (a). Find the distribution of Y.

   (b). Find the joint distribution function, $P(Y = y, K = k)$.

   (c). Find the conditional distribution function, $P(Y = y \mid K = k)$.

   (d). Find the posterior distribution function $P(K = k \mid Y = y)$.

   (e). Find the entropy of Y for $k_{out} = 20$.

   (f). Find the entropies $H(X)$, $H(Y)$, $H(Y \mid X)$, $H(X \mid Y)$.

   (g). Find the mutual information $I(K;Y)$.