EE447 Mobile Internet HW 2
The Double Spend Problem

Name: Hongjie Fang Student ID:518030910150 Email: galaxies@sjtu.edu.cn

Problem[1].

Assume that the longest chain of a blockchain acknowledges every block received. Due to the longest chain principle of the blockchain, the confirmed blocks on the chain may be discarded due to the extension of other branches, making the digital assets contained in the block potentially consumed repeatedly. This is a common double-spending problem in blockchain. To cope with this problem, the concept of "k-confirmed transaction" is proposed, which means that a transaction is officially completed only after it has received k confirmations from the blockchain.

After a transaction is confirmed, an attacker attempts to cause the transaction to be discarded by creating a branch chain. The cost per unit of time for the attacker to obtain 51% of the network-wide computing power in the blockchain is \$10,000, and the probability that 51% of the network-wide computing power successfully generates a block on the target branch per unit of time is 5% (if the attacker succeeds in generating a block per unit of time, it is considered that the branch chain adds a new block and no new block is generated on the main chain; if the attacker fails in generating a block per unit of time, it is considered that the branch chain adds no new block and no new block is generated on the main chain). If the attacker fails to generate a block per unit of time, the branch chain is considered to have not added a new block and someone else has successfully generated a block on the main chain). There is a transaction worth \$1,000,000 that is formally completed after k confirmations. What is the minimum size of k to ensure that the expected cost of the transaction to be successfully tampered with by an attacker using 51% of the computing power is higher than the amount of the transaction?

Solution. In general, we assume that the probability that the attacker successfully generates a block on the target branch per unit time is p, the cost per unit time for the attacker is c, and the target transaction is worth w. We are going to calculate the expected cost of transaction to be successfully tampered with by an attacker.

To successfully tamper with the blcokchain, the generated branch needs to have at least one more block than the branch that contains real transaction. Initially, the main branch that contains real transaction has $(k+1)$ blocks more than the attack branch, including k comfirmation blocks and one transaction block. So the attacker needs to generate $(k+2)$ extra blocks in the attack branch more than the number of blocks generated during this period in the main branch. The whole attacking process is illustrated in Fig. 1.

[1]Translated by DeepL Translator: `https://www.deepl.com/translator`

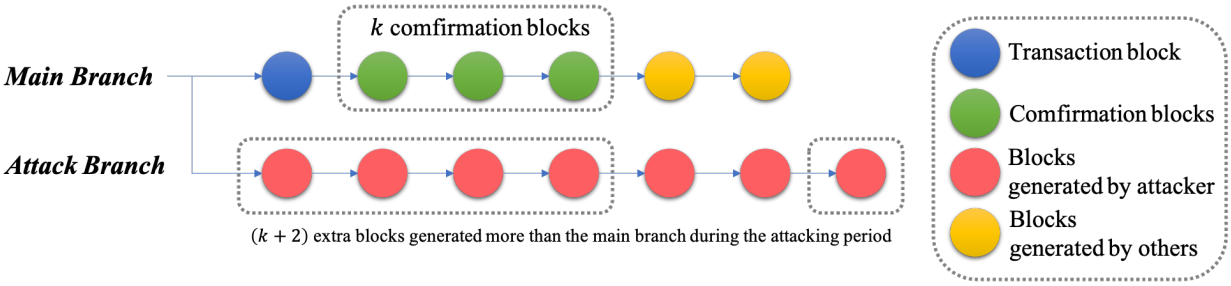

Figure 1: The whole attacking process

Apart from the extra blocks in the attack branch, we assume that in both branches, x blocks are generated during the attacking process. Therefore, totally $(k+2x+2)$ blocks are generated during the attacking process. Suppose the probability that x blocks are generated in both branches during the attacking process is $P(x)$. Hence,

$$P(x) = \binom{k+2x+2}{x} p^{k+x+2}(1-p)^x - \sum_{y=0}^{x-1} P(y) \binom{2(x-y)}{x-y} p^{x-y}(1-p)^{x-y} \quad (x \geq 1)$$

where $\binom{n}{k}$ is the combination number of parameters n and k. Initially, $P(0) = p^{k+2}$.

Now, let us explain the formula in details.

- The first term is the probability that x blocks are generated in both branches when totally $(k+2x+2)$ blocks are generated. Notice, the attacking process may already end previously because the attacking branch may already have $(k+2)$ blocks more than the making branch.
- The second term will subtract the probability that x blocks are generated in both branches when totally $(k+2x+2)$ blocks are generated, but the attacking process is already end, from the total probability calculated in the first term. Therefore, we enumerate when the process finishes, *i.e.*, the variable y. For those situations, we just need to simulate the generating process $2(x-y)$ times further, and during this time, both branches generate $(x-y)$ blocks.

Therefore, the expected cost of successfully attacking is:

$$E(cost) = \sum_{x=0}^{\infty} P(x) \cdot (k+2x+2)c$$

And we just need to find out the minimum k such that

$$E(cost) = \sum_{x=0}^{\infty} P(x) \cdot (k+2x+2)c > w$$

Since the formula includes infinite summation, and it is difficult to derive the close form solution to $P(x)$, we write a program to give the numerical solution. Since it may take long to convergence, and the value of the combination numbers may be extremely large, when $P(x_0)$ becomes too large to be computed for some x_0, we use the following form to estimate the rest part of the expected cost:

$$\left(1 - \sum_{x=0}^{x_0-1} P(x)\right) \cdot (k + 2x_0 + 2)$$

Notice that it is a quite loose lower bound estimation for expected cost, so for example, when we find out the expected cost calculated using this approach is quite close to 100, we can confidently say that it has reached 100.

```
def calc_expected_cost(k, p, c):
    '''
    Calculate the numerical value of expected cost when given k, p and c.

    Parameters
    ----------
    k, p, c: the given parameters, explained previously in the solution.

    Returns
    -------
    The numerical value of expected cost (a quite loose lower bound).
    '''
    E = 0
    P = []
    x = 0
    residual_P = 1
    while True:
        Px = comb(k + x + x + 2, x) * pow(p, k + x + 2) * pow(1 - p, x)
        if x > 0:
            for y in range(x):
                Px = Px - P[y] * comb(2 * (x - y), (x - y))
                               * pow(p, x - y) * pow(1 - p, x - y)
        if Px == np.inf:
            delta = residual_P * (k + x + x + 2) * c
            E += delta
            break
        residual_P -= Px
        P.append(Px)
        delta = Px * (k + x + x + 2) * c
        E += delta
        x += 1
    return E
```

Set $p = 0.51$, $c = 1$ (unit: \$10,000), and enumerate k then execute the program, we can list the influence of k on the expected cost as follows.

Table 1: The influence of k on the lower bound of the expected cost $\hat{E}(cost)$

k	$\hat{E}(cost)$ (unit: \$10,000)	k	$E(cost)$ (unit: \$10,000)
0	66.84764418538134	10	335.32336594674837
1	**99.3330221920333**	11	361.68535565110346
2	131.10351766835387	12	387.66995436674824
3	162.30256950229375	13	412.6743326218215
4	192.73423569681148	14	461.0166325876901
5	222.63525872853995	15	484.4196820771411
6	251.71591535341102	16	506.7371417498421
7	280.31197059901575	17	528.872260711527
8	308.0343954464944	18	549.8700375900288
9	335.32336594674837	19	570.7583380709489

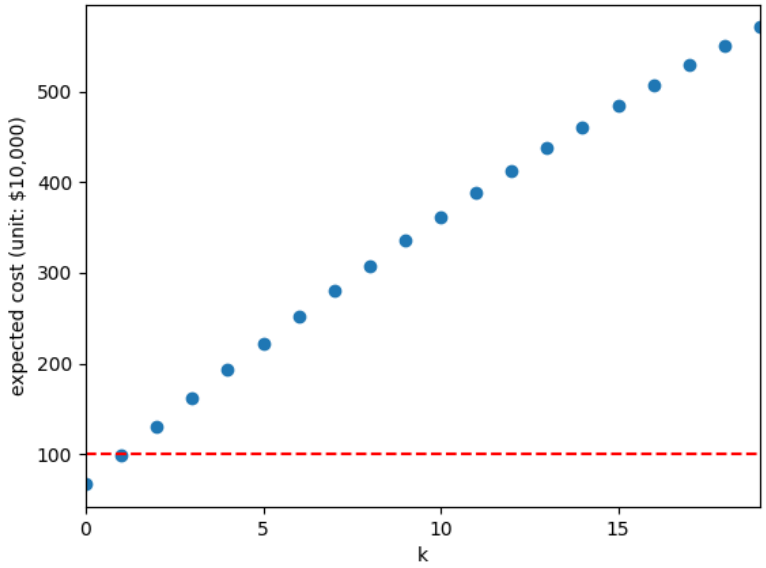

Figure 2: The influence of k on the lower bound of the expected cost $\hat{E}(cost)$

From Tab. 1 and Fig. 2 we can see that when $k = 1$, $\hat{E}(cost) \approx \$993,330$. According to the previous proposition, since we the estimate expected cost is only a loose lower bound, so we can confidently say that when $k = 1$, $E(cost) > \$1,000,000$. Hence, we choose $k = 1$ as our answer.

Now, let us interpret the answer in a more intuitive way. According to the setting of the problem, the attack branch of attacker takes 49% blocks while others takes 51% blocks when

given a certain number of blocks. From the attacker's view, the success of attack needs

$$\frac{1+2\%}{2}x - \frac{1-2\%}{2}x > k+1$$

Notice here we use $> k+1$ instead of $\geq k+2$ because in the meaning of expectation, we need continuous variables instead of discrete variables. Also, the attacker know that his/her profit must be non-negative, which indicates $x \leq 100$ according to the setting of the problem. Therefore, for attackers to succeed,

$$k < 0.02x - 1 \leq 0.02 \cdot 100 - 1 = 1$$

Hence, we need to set at least $k = 1$ to prevent attacking according to the problem settings.

In conclusion, the minimum size to ensure that the expected cost of the transaction to be successfully tampered with by an attacker using 51% of the computing power is higher than the amount of the transaction is $k = 1$. □