Notes on our BasicContract

Paul Boulanger

February 18, 2019

Let k be an element of the set K of integers (UINT64) corresponding to the account names on the EOS blockchain and let q be an element of the set Q of corresponding integers for the symbol names of the tokens. Since our smart contract constructs the primary index, which must be a UINT64 itself, of our allowance table simply by adding both the allowed spender account name and the allowed currency symbol, i.e. $k + q$, there is two possible problems. The first is a UINT64 overflow since both elements are also UINT64. This problem is not too severe and will be discussed later. The second problem is that of collisions and is potentially more serious since the primary indexes must be unique. Individually, each k and q are guaranteed to be unique, but the sum of two unique integers is not unique.

1 Probability of Collision

Given two account names k and k' from K, and two token symbols q and q' from Q, a collision is defined by $k + q = k' + q'$. By rearranging this equation,

$$k - k' = q' - q, \tag{1}$$

we can see that the condition for a collision is that there must be at least one distance between elements that is equal in each set K and Q. This condition is useful if we want to calculate the probability of collision.

To do so, we first need to calculate the number of such distances when we have N elements of one set. We label this number $D(N)$. When we consider the whole set of UINT64, we see that $D(T)$ is just the cardinality of the set, $T = \text{card(UINT64)} = 2^{64} = 18446744073709551616 \approx 1.84 \times 10^{19}$. This is rather straight forward and can be summed up that all distances are UINT64 themselves. Now, let's consider the situation where we have a small number of elements, $N << \sqrt{T}$. The calculation is again rather straight forward (see Figure 1), when $N = 1$, there is no differences, so $D(1) = 0$. When we add another element, $N = 2$, we now add 1 distance between the elements, so $D(2) = 1$. Adding another element in the set, $N = 3$, we see that we now add 2 distances, one with the first element and another with the second. We now have $D(3) = 3$. By induction we see that adding an Nth element to the set

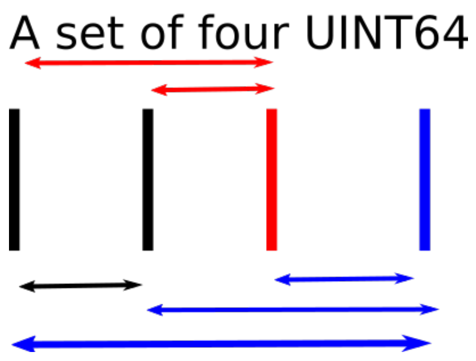

Figure 1: Illustation of the iterative increase in the number of distances when adding an element into a random set. When there is only two elements, there is only one distance (shown in black). When adding a third element, in red, this adds two new distances. A fourth random element, in blue, now adds three new distances.

increases the number of differences by $N-1$. An iterative formula to calculate the number of differences would then be:

$$D(N) = D(N-1) + N - 1. \tag{2}$$

Alternatively, we can rewrite this in a more analytical form by realising that we are only iteratively building the integer number series, which solution is known (using the Fermi trick for example). This leads to the pratical formula:

$$D(N) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} \quad \text{for} \quad N > 1. \tag{3}$$

Given a distance $d = k - k'$, the probability that the distance d is present in a set of N randomly chosen account names of K is just the number of difference in the set divided by the total number of possible differences between UINT64s, namely T.

$$P_d(N) = \frac{D(N)}{T} \tag{4}$$

Similarly, we would have the same results if we had considered a set of M randomly chosen token symbols from Q, $P_d(M) = \frac{D(M)}{T}$. The proabability that such a distance occurs in both sets simultaneously, a.k.a. a collision occurs, is

$$PC(N,M) = \frac{D(N)D(M)}{T^2} \approx \frac{N(N-1)M(M-1)}{3.39 \times 10^{38}}. \tag{5}$$

1.1 Putting the probability in perspective

Let's put some numbers into the formula to get a feeling for the order of magnitude of those probabilites. The normal use case for the allowance table is to approve a set of trusted agents (smart contracts like a DEX) that can spend a defined amount of a specified token. In general, a user would have maybe $O(10)$ such spenders and maybe a handful of tokens, also $O(10)$. The probability of collision would thus be:

$$PC(10, 10) \approx \frac{10 \times 9 \times 10 \times 9}{3.39 \times 10^{38}} = 2.4 \times 10^{-35}. \tag{6}$$

This is an extremelly low chance of collision. Since the probability of winning the Powerball jackpot is about one in 292 millions, that is to say the probability is about 3.4×10^{-9}. So a collision as the same probability as winning about **four times the Powerball jackpot in a row**. That is to say, it is so small that it should not be a problem.

To get a feeling of how this probability scales (as N^2 and M^2), let's imagine the absurd scenario where someone allows everybody on Earth (in this scenario everybody is on EOS), $N = 7 \times 10^9$, to spend all possible tokens which are on EOs right now, $M = O(1000)$. Let's note that this is close to the limit $N \approx \sqrt{T}$ so you can expect some deviation. With those caveats, the probaility is estimated to be:

$$PC(7 \times 10^9, 1000) \approx 1.45 \times 10^{-10}. \tag{7}$$

This is about like **winning the Powerball jackpot**. So even in this ludicrous example, the probability of collision is so low that it is unlikely to become a problem.

Finally, let us note that the problem of collision as a long history in the cryptocurrency world. Bitcoin addresses function on a random hashing function which also can lead to collisions. Finding a collision in the P2PKH would spell the end of this way, which is the gold standard right now, of generating addresses. [1] The probability of collisions in P2PKH is roughly the same order of magnitude as the one in our contract.

[1] See this article for example