Scribe 4: Rho Attack

Instructor: Viet Tung Hoang

Disclaimer: These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.

Motivation. Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a hash function and let $N = 2^n$. Suppose that we want to find a collision of $H$. To speed up the running time, we want to run a collision-finding attack on every processor of a GPU. However, since those processors have a limited shared memory, it is crucial that the attack must use very little memory, preferably $O(1)$ memory. This rules out the naive birthday attack, since that requires $\Omega(\sqrt{N})$ memory.

The rho method. Consider the following process. Initially, we start with a random string $x_0 \leftarrow \{0, 1\}^n$, and then iterate $x_1 \leftarrow H(x_0), x_2 \leftarrow H(x_1)$, and so on. Since these strings take value from a finite set $\{0, 1\}^n$, eventually there must be $i < j$ such that $x_i = x_j$. But then $x_{i+1} = H(x_i)$ and $x_{j+1} = H(x_j)$ must be the same. In addition, $x_{i+1} = H(x_{i+1})$ and $x_{j+2} = H(x_{j+1})$ must also be the same, and so on. In other words, for every $k \geq 0$, we must have $x_{i+k} = x_{j+k}$. See Figure 4.1 for an illustration. Pictorially, the sequence $x_0, x_1, \cdots$ form a rho shape: it takes us some $r$ steps to enter a cycle of length $\ell$, where $r = 3$ and $\ell = 6$ in the example of Figure 4.1. If $r \geq 1$ then $x_{r-1}$ and $x_{r+\ell-1}$ form a collision of $H$, since $x_{r-1} \neq x_{r+\ell-1}$, yet $H(x_{r-1}) = x_r = H(x_{r+\ell-1}).$

![Figure 4.1: Illustration of the rho shape. Here $x_3 = x_9$, and thus $x_{3+k} = x_{9+k}$ for every $k \geq 0$.](image)

Note that the rho method above may fail to generate a collision if $r = 0$, as illustrated in Figure 4.2. In this case, the rho shape degenerates into a cycle.

Note that if we model $H$ as a random oracle then $x_0, x_1, \cdots$ can be modeled as independent, uniformly random strings (until repetition happens at step $L = r + \ell$). Then with high probability, the repetition will happen within $\sqrt{2N}$ steps—recall the Birthday Paradox—and thus it’s very likely that $L = O(\sqrt{N})$.

Now, we want to use the rho method above to find a collision. However, there are several daunting obstacles.
Figure 4.2: A degenerate case where the rho method fails to generate a collision.

First, recall that we have only $O(1)$ memory, so we can only store just a few strings $x_i$ in memory at a time. Moreover, we don’t want to run $\Theta(\sqrt{N})$ steps. Since collision happens after $L$ steps, we want to terminate after $O(L)$ steps, although we don’t know what $L$ is. The attack consists of two steps: (i) detecting the presence of a cycle, and (ii) finding collision, both using $O(1)$ memory and $O(L)$ time.

**Floyd’s cycle detection.** Note that for each choice of $x_0$, there is a unique number $m \leq L$ such that $x_{2m} = m$. (For the example in Figure 4.1, $m = 6$.) To see why, note that $x_{2m} = m$ if and only if (i) $m \geq r$ (meaning that you should at least enter the cycle to have repetition), and (2) $m$ is a multiple of $\ell$ (meaning that the gap $m$ between the two positions $x_m$ and $x_{2m}$ should be a multiple of the cycle length). However, there is exactly one number among $\ell$ consecutive numbers $r, r+1, \ldots, L = r + \ell - 1$ that is divisible by $\ell$.

Floyd’s algorithm aims to find $x_m$ from $x_0$ after $O(L)$ steps, using $O(1)$ memory. To have an intuition of the algorithm, imagine a running race between a hare and a tortoise along the rho shape, both starting at the initial point $x_0$. At each iteration the hare can run 2 steps, whereas the tortoise can only run 1 step. So at the $k$-th iteration, the tortoise is at position $x_k$, whereas the hare is at position $y_k = x_{2k}$. Hence the next time the two animals meet, this is at position $x_m = x_{2m}$.

Formally, given $x_0$, the algorithm initializes $y_0 \leftarrow x_0$ and proceeds as follows. At each step $k$, the algorithm will keep track of just two strings $(x_k, y_k)$, and terminate if $x_k = y_k$. To move from step $k$ to step $k + 1$, we compute $x_{k+1} \leftarrow H(x_k)$ and $y_{k+1} = H(H(y_k))$. Note that $y_k = x_{2k}$ for every $k \geq 0$. Hence the memory usage is just $O(1)$ and the algorithm stops at step $m$, returning $x_m$.

**Collision finding.** Now, from $(x_0, x_m)$, we want to find the collision $(x_{r-1}, x_{t+r-1})$ using $O(1)$ memory and $O(L)$ time. (In Figure 4.1, it means that we want to find $(x_2, x_8)$ from $(x_0, x_6)$.)

To have an intuition of our method, imagine that we have two tortoises at positions $x_0$ and $x_m$, running along the rho shape. At each iteration, each tortoise can only move one step, so at the first iteration, they will be at positions $x_1$ and $x_{m+1}$ respectively, and so on. We claim that when the two tortoises first meet, they will be at the position $x_r$. (In Figure 4.1, you can see that at the third iteration, the two tortoises will meet at $x_3$.) To see why, note that at the $r$-th iteration, the two tortoises will be at positions $x_r$ and $x_{m+r}$ respectively. Since $m$ is a multiple of $\ell$, this means that the position $x_{m+r}$ is the same as $x_r$. So intuitively, to find the collision, we just need to keep track of the tortoises’ current positions, and stop them right before they hit each other.
Formally, in iteration $k$, we keep track of $(x_k, x_{m+k})$ and terminate if $H(x_k) = H(x_{m+k})$, and thus the memory usage is $O(1)$. To move from iteration $k$ to iteration $k+1$, we update $x_{k+1} \leftarrow H(x_k)$ and $x_{m+k+1} \leftarrow H(x_{m+k})$. Thus we will terminate after $r$ steps, and the running time is $O(L)$.

**Remark.** It is instructive to see what happens when we apply the algorithms above in the degenerate case, where the rho method generates a cycle, instead of a rho shape. In that case, $r = 0$ and $m = \ell$. (In the example of Figure 4.2, we have $m = \ell = 6$.) When we apply the Floyd’s algorithm, we’ll get back $x_m = x_0$. Thus when we try to find a collision, our two tortoises will start from the same position $x_0$, and we’ll terminate immediately, since they will surely hit each other in the next iteration.