The Birthday Paradox and Cryptography

Receiving chocolates on birthdays!

Probability can often seem counter-intuitive at first. Consider a delightful and nostalgic situation from our childhood when we students distributed chocolates on our birthdays . A particular child with a sweet tooth might wonder, what is the probability that I receive 2 chocolates today, or in other words, 2 people share a birthday. Surprisingly, the birthday paradox mathematically shows that in a group of just 23 students, there's actually over a 50% chance that at least two students will share the same birthday. In a class of 40 students, the probability of a shared birthday becomes 89.1%.Let's dive into how!

The math behind it

To determine the probability that at least two people in a group of 23 share the same birthday, we first calculate the probability that no two people share the same birthday, denoted as $P(A)$. We start by defining the total number of possible birthday assignments for 23 people, denoted as V_t, which is 365^{23}, because each of the 23 people can have any of the 365 birthdays, and repetition is allowed. Next, we calculate the number of favorable outcomes, V_{nr}, where no two people share the same birthday. This can be expressed as a permutation, where the first person has 365 choices, the second person has 364, and so on, leading to the formula $V_{nr} = \frac{365!}{(365-23)!}$. The probability $P(A)$ is then the ratio of these favorable outcomes to the total outcomes, $P(A) = \frac{V_{nr}}{V_t} = \frac{\frac{365!}{(365-23)!}}{365^{23}} \approx 0.492703$. The desired probability that at least two people share a birthday, $P(B)$, is the complement of $P(A)$, which gives $P(B) = 1 - P(A) \approx 1 - 0.492703 \approx 0.507297$. This result indicates that in a group of 23 people, there is approximately a 50.73% chance that at least two people will share the same birthday. Even if the distribution of actual birthdays is not perfectly uniform (e.g., some days might have more births than others due to cultural or seasonal factors), this non-uniformity is generally not significant enough to drastically change the number of people required to have at least a 50% chance of sharing a birthday. Moreover, The Pigeonhole Principle states that if you have more items than containers and you try to place each item into a container, at least one container must contain more than one item. Applied to the birthday problem, this principle implies that if there are more people than possible unique birthdays, at least two people must share a birthday. Specifically, with 367 people and only 366 possible birthdays (including February 29th in a leap year), there is a 100% probability that at least one pair of people will share the same birthday.

Hash Functions and Birthday Collision Attacks

Hash functions are algorithms that convert input data into a fixed-size hash value. The goal of a birthday collision attack is to find two distinct inputs, x_1 and x_2, such that they produce the same hash value, i.e., $f(x_1) = f(x_2)$. Due to the birthday paradox, finding such a collision is more efficient than one might intuitively expect. Specifically, if a hash function f produces H possible outputs, you only need about $1.25\sqrt{H}$ different inputs to have a 50% chance of finding a collision. To quantify

this, the probability $p(n; H)$ of finding at least one collision after choosing n values from H possible outputs can be approximated by $p(n; H) \approx 1 - e^{-\frac{n^2}{2H}}$. To determine the number of values $n(p; H)$ needed to achieve a collision probability of at least p, the formula is $n(p; H) \approx \sqrt{2H \ln \frac{1}{1-p}}$, which for a 50% chance simplifies to $n(0.5; H) \approx 1.1774\sqrt{H}$. The expected number of values required to find the first collision, denoted $Q(H)$, is approximately $Q(H) \approx \sqrt{\frac{\pi}{2}H}$. For an 8-bit hash function, which has $H = 256$ possible hash values, this implies that approximately $n(0.5; 256) \approx 1.1774\sqrt{256} \approx 18$ different inputs are needed to achieve a 50% chance of finding a collision.

Digital signatures are vulnerable to birthday attacks, particularly chosen-prefix collision attacks. In such an attack, a malicious actor, Mallory, creates two different messages, m and m', where m is a legitimate contract and m' is fraudulent. By generating numerous variations of both m and m', Mallory aims to find a pair of messages that hash to the same value, i.e., $f(m) = f(m')$. Once a collision is found, Mallory can get Bob to sign the fair contract m, then use that signature on the fraudulent m'. The probability of finding such a collision is about $2^{(n/2)+1}$ hashes for a 50% chance, where 2^n is the number of possible hash values.To mitigate such attacks, hash functions should have sufficiently long output lengths to make the birthday attack infeasible. Additionally, signers should make random changes to the document before signing and keep a copy of the signed document for verification.Pollard's rho algorithm, a technique used for discrete logarithms, also leverages birthday collisions. It finds collisions in a sequence of values by exploiting the birthday paradox, significantly speeding up the computation compared to brute-force methods.