Birthday Attack

Ronny Yang, Sonika Verma, Tashrif Sanil

December 07, 2020

Overview

The Birthday Paradox is quite a well-known phenomenon within the realm of simple mathematics and probability. The question is proposed: how many people need to be in a room for any two to share the same birthday? Although not an actual paradox, the Birthday Paradox is named as such, as the majority of people think relative to a single person. Using simple properties of probability, for there to be a 50% chance that a single person shares the same birthday with someone in the room, there needs to be a group of 253 people. However, considering any two people in the room sharing a birthday, only 23 people are needed to have a probability higher than 50%.

Similarly, the Birthday Attack is a brute-force attack within cryptography and information security that uses the same mathematics to focus on what is referred to as collisions in a system. The success rate of the attack is based on how likely an attacker is able to find collisions in a given system via trying multiple inputs to compute for the same output. For a cryptography system using a hash table to encode inputs, this would equate to the same as finding any two inputs that share the same output hash value. Should an attacker be able to find enough collisions, using reverse-engineering the system can be compromised as the encryption mappings become known to the attacker. Even with a single collision, a system can be compromised as the attacker can use it to perform a various amount of fraudulent actions and/or man-in-the-middle attacks.

Terminology

Birthday Paradox	A problem that asks the proposes the probability of two people in the same room having the same birthday, and how the probability changes when considering the number of people in the room
Pigeonhole Principle	Given n+1 pigeons and n pigeonholes, you can conclude that at least one pigeonhole has more than one pigeon.
Brute Force Attack	A method of attacking a system that is based on systematically trying multiple or all possible inputs to achieve a desired output.
Hash Function	A system used to map inputs as keys to a fixed-length output hash code used to index data held in [hash] tables. Often used to store passwords in an encrypted and secure fashion.
Plain text	Sequence of unencoded, unformatted Unicode characters containing only text.
Collision	Duplication of output values within the same system for unique different values. Given unique keys x, y in a system F, there is a collision if F(x) = F(y) happens when a function is not one-way or bijective.
One-Way	A property of hash functions implying it is nearly impossible to compute to reverse engineer for a key x when given a hash value h where h = H(x)
MD5	Message-Digest Algorithm; A widely used hash-function that produces a 128-bit hash value to a given input. No longer actively used as it suffers from many vulnerabilities.
Digital Signatures	A method to reproduce physical signatures on technology, as well as provide additional security by verifying senders' identity and provide assurance for the integrity of the document.

Birthday Paradox

Consider the setup of the Birthday Paradox introduced earlier. Assume leap years are not considered and assume an even birth distribution. Regardless of the interpretation of the question, by the Pigeonhole Principle, it can be concluded that given 366 people there is a 100% probability that one of them will share a birthday as there are only 365 possible days in a year.

For simplicity, the probability that one person shares a birthday - denoted $P(n)$ - can also be denoted as $1 - P(\neg n)$ where $P(\neg n)$ is the probability of not sharing a birthday. This is due to the fact that $P(n)$ and $P(\neg n)$ are mutually disjoint sets.

If one were to make the common mistake and select one person as a chosen candidate and look for a different person with a matching birthday it can be concluded that for there to be a 50% chance that someone shares the same birthday as the candidate, there must be 253 people in the room. This can be seen through the following calculations:

The probability that a person does not share the same birthday with the candidate is represented as $\frac{364}{365}$
The probability that two people do not share a birthday with the candidate can be represented as $\frac{364}{365} \times \frac{364}{365}$

Then, the probability that a room of n people do not share a birthday with the candidate can be computed as $(\frac{364}{365})^n$
Following, the probability of a person sharing a birthday with the candidate can be computed as:

$$P(shared\,birthday)1 - P(not\,shared\,birthday)$$
$$0.50047715403 \approx 1 - \left(\frac{364}{365}\right)^{253}$$

Now, consider any two people in the room sharing a birthday, instead of sharing one with a chosen candidate. The Birthday Paradox states that only 23 people are needed to have a probability higher than 50%. This follows as a room full of 23 pairs will result in 253 potential pairs, which is the generalized version of the earlier problem and can be seen below.

A person alone in a room is guaranteed to not share a birthday with anyone else and is represented as $\frac{365}{365}$
Adding another person, the probability the two people in the room do not share the same birthday is represented as $\frac{364}{365}$
Adding a third, the probability that the third person in the room does not share a birthday with the previous two is represented as $\frac{363}{365}$.
This pattern can be generalized with more people added as $\frac{362}{365}, \frac{361}{365}, \frac{360}{365}...$
Multiplying these values together will result in the probability that no two people in the room share the same birthday

Consider a room of 23 people. The probability that no two persons will share the same birthday is computed as:

$$\begin{aligned} P &= \left(\frac{365}{365}\right) \cdot \left(\frac{364}{365}\right) \cdot \left(\frac{363}{365}\right) \cdot \left(\frac{362}{365}\right) \cdot ... \cdot \left(\frac{343}{365}\right) \\ &\approx 0.492702765676 \end{aligned}$$

Then, the probability two people of the 23 in a room having the same birthday is computed as:

$$\begin{aligned} P(shared\,birthday) &= 1 - P(\neg shared\,birthday) \\ &= 1 - 0.492702765676 \\ &\approx 0.507 \\ &\approx 50\% \end{aligned}$$

Generalization

The Birthday Paradox - as demonstrated above - can also be written as a general formula to calculate the probability that all people in the entire room do not share a same birthday:

$$\begin{aligned} P(\neg n) &= \left(\frac{365}{365}\right) \cdot \left(\frac{364}{365}\right) \cdot \left(\frac{363}{365}\right) \cdot \left(\frac{362}{365}\right) \cdot ... \cdot \left(\frac{365 - \neg n + 1}{365}\right) \\ &= \frac{365 \cdot 364 \cdot ... \cdot (365 - \neg n + 1)}{365^n} \\ &= \frac{365!}{365^{\neg n} \cdot (365 - \neg n)!} \end{aligned}$$

Which can then be used to compute the probability two people sharing a birthday as:

$$P(n) = 1 - P(\neg n)$$

Birthday Attack

The Birthday Attack uses the formula created in the Birthday Paradox, but further generalized by rewriting the number of days as (k) and the number of people as (n).

$$\begin{aligned} P(n, k) &= 1 - \frac{k!}{k^n \cdot (k - n)!} \\ &= 1 - \left(\frac{k}{k}\right) \cdot \left(\frac{k-1}{k}\right) \cdot \left(\frac{k-2}{k}\right) \cdot \left(\frac{k-3}{k}\right) \cdot ... \cdot \left(\frac{k-n+1}{k}\right) \end{aligned}$$

Using Pigeonhole Principle, it is known that for any function/system that can take more inputs than the fixed number of outputs, collisions must occur. The formula as stated above considers a similar problem to the Birthday Paradox, but generalizes for n attempts in a system with k possible outputs. This is known to calculate the probability $P(n, k)$ to find a collision for the Birthday Attack.

Simplification

Given the generalized formula above, it can be re-written as the following:

$$P(n, k) = 1 - 1 \cdot \left(1 - \frac{1}{k}\right) \cdot \left(1 - \frac{2}{k}\right) \cdot \left(1 - \frac{3}{k}\right) \cdot ... \cdot \left(1 - \frac{n}{k}\right)$$

Consider the known inequality $e^{-\frac{1}{x}} \geq 1 - \frac{1}{x}$, then:

$$\begin{aligned}
P(n,k) &\geq 1 - 1 \cdot \left(e^{-\frac{1}{k}}\right) \cdot \left(e^{-\frac{2}{k}}\right) \cdot \left(e^{-\frac{3}{k}}\right) \cdot \ldots \cdot \left(e^{-\frac{n-1}{k}}\right) \\
P(n,k) &\approx 1 - e^{-\frac{n(n-1)}{2k}} \\
&\approx 1 - e^{\frac{-n^2}{2k}}, \text{ for large } n
\end{aligned}$$

The equation can further be rearranged to consider how many inputs n will be needed for a function with a fixed output k and varying probability p:

$$\begin{aligned}
P &= 1 - e^{-\frac{n^2}{2k}} \\
e^{-\frac{n^2}{2k}} &= 1 - p \\
-\frac{n^2}{2k} &= ln(1-p) \\
\frac{n^2}{2k} &= ln\left(\frac{1}{1-p}\right) \\
n &= \sqrt{2k \cdot ln\left(\frac{1}{1-p}\right)}
\end{aligned}$$

Example

For a system with 256 possible outputs if an attacker want a collision to occur with 75% probability:

$$n = \sqrt{2(256) \times ln\left(\frac{1}{1-0.75}\right)} \approx 27$$

So it will take approximately 27 attempts for an attacker to find a collision with 75% probability.

Collisions

The Birthday Attack provides a way to solve for how many inputs an attacker will need to try to guarantee a certain percentage of finding a collision in the system. This can be done via a brute-force tactic of trying different inputs through the system up until the same result is seen as an output for two unique inputs. Those two inputs mapping to the same input would be considered a collision in the system. Collisions in any system meant to be secure prove to be a large vulnerability of the system based on how the collision can be abused by an attacker without the system or users being able to detect it.

> Consider a system that matches a given username to a user permission levels. The system is claimed to be secure as no outside user knows how the mapping is done or what functions are applied to the input for it to be converted to the output (is a black-box system to users). However, an attacker does not need to know how the system works to exploit a Birthday Attack vulnerability. Suppose the attacker is able to brute-force test a few values and is able to record the following information:
>
> $$\begin{aligned} F(0) &= 0 \\ F(3) &= 3 \\ F(4) &= 4 \\ F(-3) &= 3 \end{aligned}$$
>
> Using the above data, the attacker can conclude there is a collision for the usernames 3 and -3. The attacker can then compromise the system by being able to login with a fraudulent or invalid username and replicate the behaviour from a valid username.

Collisions are a significant vulnerability is such systems. Consider a hash code that maps keys to values, then, the Birthday Attack can determine the probability in which they can find a collision within n attempted keys for the code with k possible hash values.
Note, collisions happen with functions that are surjective, and allow unique keys to be mapped to the same value.

Applications of Birthday Attack

As a brute-force attack, the Birthday Attack is applicable to any system that has more inputs than outputs it can map to. This is observed by properties of the Pigeonhole Principle applied to surjective functions.

Breaking MD5

MD5 - also known as Message Digest Algorithm - is a widely used one-way hashing algorithm that produces a fixed length 128-bit hash value to a given a message of any length. The returned hash-value can then be used for authentication and verification of the initial hashed message.
However, MD5 has seen been deprecated to no longer be used as a cryptographic hashing algorithm as it is vulnerable to the Birthday Attack. Finding collisions compromises the system's security as the attacker can make inferences on how the hash is implemented, and can perform authenticated actions with invalid inputs as the resulting hash's are the same, and produce the same behaviour.

MD5 will always produce a fixed-length 128-bits hash to any given input (regardless of length). Then,

$$\text{No. of possible hashes } = 2^{128} = 3.4 \times 10^{38}$$

Consider a website that accepts passwords of length 20 with 94 acceptable characters that can be used in the password. The website then uses MD5 to hash and store the passwords on a database, with the usernames stored as plain text. Say an attacker - Mallory has been able to obtain the database containing usernames and hashed passwords.
Mallory knows that each password can have 94 possibilities for each character of the 20 characters so in total, so Mallory can determine:

$$\text{No. of possible passwords } = 94^{20} = 2.9 \times 10^{39}$$

Note that if Mallory was to try all possible passwords of which there would be 94^{20} possibilities, it would take approximately 1.4×10^{25} years to generate all computations. This would prove to be a very costly action for Mallory both for the time consumed and computational space to store and try all of the different hashes.

However, Mallory can exploit the Birthday Attack to reduce the computational time by looking for collisions. If Mallory can find a collision, she would be able to use the password generating the matching hash in the data in storage to login to the website as the associated authenticated user.
Suppose Mallory wants a 99% chance of finding a collision ($p = 0.99$) where k is the total number of possible hashes ($k = 2^{128}$).
Then, Mallory can find the number of input passwords n to generate along with their equivalent hashed to be evaluated.

$$\begin{aligned} n &= \sqrt{2k \cdot ln\left(\frac{1}{1-p}\right)} \\ &= \sqrt{2(2^{128}) \cdot ln\left(\frac{1}{1-0.99}\right)} \\ &\approx 5.98 \times 10^{19} \end{aligned}$$

Following, Mallory can conclude that she only needs to generate 5.98×10^{19} possible passwords and their respective hash to have a 99% chance of finding a collision (a generated hash that would match a valid hashed password in the website database).
5.98×10^{19} is less than one trillionth of 94^{20}, and Mallory would be able to compute this within a few (about 8) months on a high end PC. If Mallory only needed a 75% or 50% chance of finding a collision, the computational time would further be reduced.

Then, if Mallory is able to find a hash collision, Mallory has successfully compromised the MD5 hash system used on the website.

Digital Signature Susceptibility

Digital Signatures are a mathematical technique used to verify and validate the authenticity and integrity of a digital message or document. Used to substitute a handwritten signature, digital signatures aim to offer a strong reason to believe the signed document is secure and sent by an authenticated and signed user, where the contents of the message have not been altered or tampered with.
Digital Signatures are often used to provide a safe and authenticated channel of communication, and avoid impersonation of users via an assurance of the signature that provides evidence of the origin of the document and signed identity of the sender.

Suppose Mallory is trying to get a user - Alice - to sign a fraudulent contract. Alice trusts the cryptographic hashes that are advertised to be secure and her own signature that she would sign with.
Suppose the document is protected with a fixed-length 128-bit hash which will result in:

$$\text{No. of possible hashes } = 2^{128} = 3.4 \times 10^{38}$$

As seen in MD5, Mallory knows applying the Birthday Attack she only needs 2.2×10^{19} attempts before having 50% chance of a collision, or 3.3×10^{19} for a 75% chance of a collision.

Mallory can then start to create minor changes in the valid contract such as adding spaces, extra punctuation or formatting that would result in the contract content remaining the same, but a varying hash value as the unicode continues to change. Similarly, Mallory can apply small changes for the fraudulent contract.
Suppose Mallory writes some simple code to add minor changes to alter both valid and fraudulent contracts creating 3.3×10^{19} real contracts, and 3.3×10^{19} fraudulent ones. Then by application of the Birthday Attack, Mallory can ensure there is a 75% chance that a valid and fraudulent contract will result in the same hash.

Mallory then presents Alice with the altered valid contract that has a colliding hash with the fraudulent contract. Once Alice signs the valid contract, Mallory can easily transfer the digital signature to the fraudulent contract as they have the same hash value. Mallory has then obtained a fraudulent contract that is signed by an user to confirm validity of the document at the signed user's expense.

Then, since the digital signature is designed to confirm authenticity and intangibility of the document, Mallory has successfully been able to impersonate an authenticated user.

Visualizing Birthday Attacks

Visualization of Birthday Attacks can be done by graphing collisions of any given hash function. Birthday Attacks rely solely on the fact that hash functions have more inputs then they have outputs, and thus any hash function can be used to visualize the attack.

The Pearson Hash

Consider the following hash function:

```
table_of_values = list(range(0, 256))
shuffle(table_of_values)

def pearson_hash(message: str, table) -> int:
    hash = len(message) % 256
    for i in message:
        hash = table[hash ^ ord(i)]
    return hash
```

The Pearson Hash is a 8-bit hash function with 256 possible hash values between 0 and 255. The function takes an input as well as a 256 length table of values between 0 and 255 in any order of the users choosing. Depending on the input table, the hash value of the same input may change. Note that the Pearson Hash is not meant to be a cryptographic hash function, but as it shares the same property of having infinite inputs and limited outputs it can be used to visualize the Birthday Attack nonetheless.

Sample Collision

Consider the following input and corresponding output from the hash:

```
table_of_values = list(range(0, 256))

def pearson_hash(message: str, table) -> int:
    """Pearson Hashing Function"""
    hash = len(message) % 256
    for i in message:
        hash = table[hash ^ ord(i)]
    return hash

if __name__ == '__main__':
    hashes = dict()
    inputs = ["Woah", "isn't", "this", "such", "a", "cool",
              "example?", "Wow,", "I", "wonder", "if", "it",
              "is", "the", "coolest", "example", "that", "has",
              "ever", "been", "made", "?", "bananas"]

    for i in inputs:
            hashes[i] = pearson_hash(i, table_of_values)

    print(hashes)
```

The given example uses a simple table of values [0,1,2,3,...., 254,255], alongside the inputs:
["Woah", "isn't", "this", "such", "a", "cool", "example?", "Wow,", "I", "wonder", "if", "it", "is", "the", "coolest", "example", "that", "has", "ever", "been", "made", "?", "bananas"]

Each input is passed through the Pearson hash and is mapped to its associated hash value:
['Woah': 53, 'isn't': 34, 'this': 2, 'such': 9, 'a': 96, 'cool': 11, 'example?': 95, 'Wow,': 103, 'I': 72, 'wonder': 3, 'if': 13, 'it': 31, 'is': 24, 'the': 122, 'coolest': 106, 'example': 111, 'that': 13, 'has': 121, 'ever': 0, 'been': 8, 'made': 9, '?': 62, 'bananas': 119]

Here, a collision can be seen for the hash value 9 between the inputs "such" and "made", as well as for the hash value 13 between the inputs "if" and "that".

Scaling Collisions

In the previous example, a collision was found after 17 attempted inputs. The same process can then be repeated at a much larger scale to visualize Birthday Attacks more thoroughly. Consider the following code:

```
import random
import string
from random import shuffle
import matplotlib.pyplot as plt
from collections import Counter
table_of_values = list(range(0, 256))
shuffle(table_of_values)
```

```python
def pearson_hash(message: str, table) -> int:
    """Pearson Hashing Function"""
    hash = len(message) % 256
    for i in message:
        hash = table[hash ^ ord(i)]
    return hash


def generate_string(length):
    """Random String Generator"""
    letters = string.ascii_lowercase
    result = ''.join(random.choice(letters) for i in range(length))
    return result

if __name__ == '__main__':
    attempts = 0
    hashes = dict()
    results = []
    collisions = 10000000
    string_length = 10
    while len(results) < collisions:
        hash_input = generate_string(string_length)
        while hash_input in hashes.keys():
            # Generates a random string and confirms it is unique
            hash_input = generate_string(string_length)
        output = pearson_hash(hash_input, table_of_values)
        if output in hashes.values():
            # Collision is found
            results.append(attempts)
            hashes = dict()
            attempts = 0
        else:
            # Collision is not found
            hashes[hash_input] = output
            attempts += 1

    """Graphing the results"""
    plt.bar(Counter(results).keys(), Counter(results).values())
    plt.ylabel('Number of Collisions')
    plt.xlabel('Inputs Till Collision')
    plt.title("Distribution of Collisions for the Pearson Hash")
    plt.show()
```

In the code above, unique and random ten length strings are generated and hashed until a collision is found. Once found, the number of inputs hashed till a collision was found is recorded. This process is repeated ten million times and is graphed below.

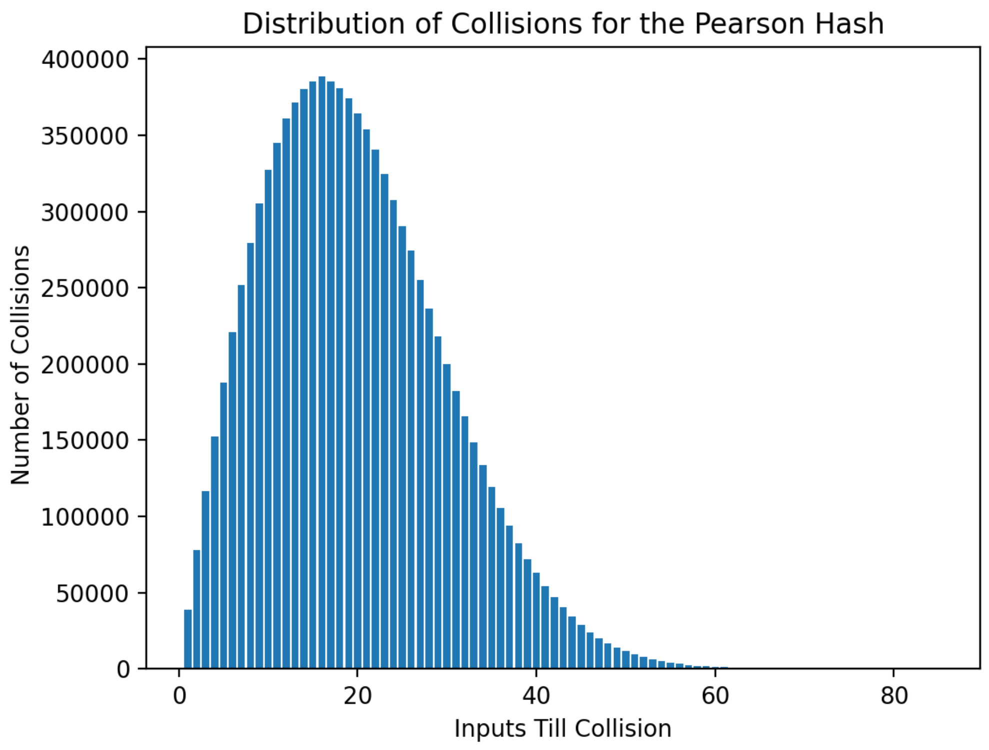

Figure 1: The graph maps the number of collisions found against the number of inputs tried. For example, about 350,000 collisions of the 10 million found occurred after 20 inputs were hashed.

Although generated solely through the collisions of the Pearson Hash, the graph approximates the function of the Birthday Attack remarkably well. Viewing the original function

$$P = 1 - e^{-\frac{n^2}{2k}}$$

as a cumulative distribution function, its probability distribution function can be found by taking it's derivative:

$$P' = \frac{ne^{-\frac{n^2}{2k}}}{k}$$

Using the value ($k = 2^8 = 256$) for the Pearson Hash the formula is finalized alongside its corresponding graph:

$$P' = \frac{ne^{-\frac{n^2}{512}}}{256}$$

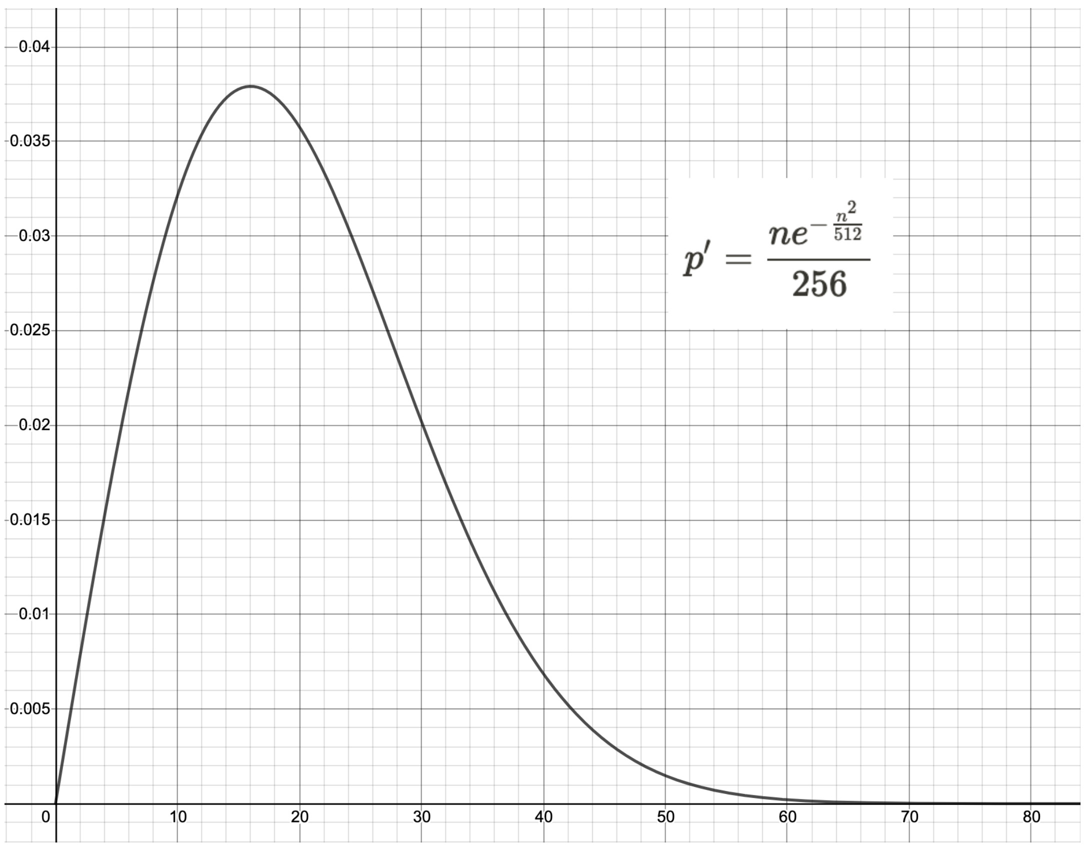

Figure 2: The Probability Distribution Function of P

When graphed, a clear correlation can be seen between the collision generated graph and of the probability distribution function for P (p'). An important point to remember is that the collision generated graph did not involve the function P derived from the Birthday Attack, but rather only considered the repeated collisions found within the Pearson Hash. Similar graphs can be recreated with different hash functions with enough collisions.

Considering different k-values given different hash functions, it is possible to find equivalent distributions without computing millions of collisions. Comparing the collision generated graphs to the the graph of p' it can be observed that it will always follow a similar distribution. This demonstrates that the function derived for the Birthday Attack actively applies to any hash functions, and can be used to provide a clear visualization for the math previously derived.

Defense & Prevention

Protection against the Birthday Attack is the same as protecting against any brute-force attack, and can be achieved in a similar mathematical sense by increasing the output length of the hash function. The larger the codomain is, the less likely there are to be collisions. However, this is still not foolproof, so it is rather advised to increase the length(s) of the hash function (usually by a factor of 2). This way, the space the hash function maps to becomes exponentially large and it is no longer computationally feasible to use a brute-force method to attack the system. Following the same method, it also becomes increasingly hard/infeasible to reverse engineer the hash values, making the hash one-way. A good example to regard is SHA-2 which commonly produces a 256 or 512-bit hash.

It is also advised to create hash functions that are collision free so no two distinct inputs map to the same output [consider SHA-2]. This is often achieved by making the function injective or bijective (based on acceptable input values) so each unique key is mapped to at most one hash code if the possible inputs is comparable to the size of the possible outputs.
Hashes are also often salted - where some random data (salt) is used as additional input to the given string before it is converted into a hash value. Other hashes are 'catastrophic' where small changes in the input will result in big changes in the resulting hash (also known as the butterfly effect), which can make it harder to find similar hashes.

However, the best defense against vulnerabilities such as Birthday Attacks continues to be awareness of the problem and available solutions.

Citations

Gupta, Ganesh. “What Is Birthday Attack??” Research Gate, Feb. 2015, doi:10.13140/2.1.4915.7443.

Rivest, R. (April 1992). "[Step 4. Process Message in 16-Word Blocks](https://tools.ietf.org/html/rfc1321 section-3.4)". The MD5 Message-Digest Algorithm

RSA Laboratories. 2.2.2 What Is a Digital Signature and What Is Authentication? web.archive.org/web/20040913080209/www.rsasecurity.com/rsalabs/node.asp?id=2182.