Thomas Hart
CS465
Project 2 – Hash Attack

Hashes

In computer science, a hash function takes an arbitrary amount of data (or bits) as input and returns a random string of bits. For example, I could take all the 1's and 0's that make up this digital report, put it through a hash function, and receive back a relatively smaller amount of random 1's and 0's. A hash function is useful because no two inputs produce the same output. Let's explore why this is useful with some examples.

Integrity

Attackers with the appropriate knowledge and skill can manipulate the contents of data on its way to a destination. This is a problem of integrity. For example, if I send this document over the internet, how does the intended receiver know that its contents have not been changed by an attacker? There must be a way to detect this.

Hash functions, in theory, ensure that messages are what they're supposed to be. If each input to a hash function provides a unique output (and that input always produces the same unique output), the 1's and 0's returned by the function act as a certificate, or "digital signature". This report – and ONLY this report – should produce a certain output when put through a hash function. In practice, I can save that output, send my report, have the report sent through the hash function after it's been delivered, and compare the two results. If they are the same, we know my report was not tampered with on its way to the receiver. This method ensures the integrity and authenticity of data.

Ease-of-Use and Human-Readability

Hashes are convenient because they are quick and easy to implement and can be applied to inputs of any size. They are also easy for humans to understand. A hash function returns a series of bits that can be printed to a computer screen as a few random letters and numbers. This is easy for a person to compare two outputs and verify the authenticity of messages or other forms of data.

Hash Attack

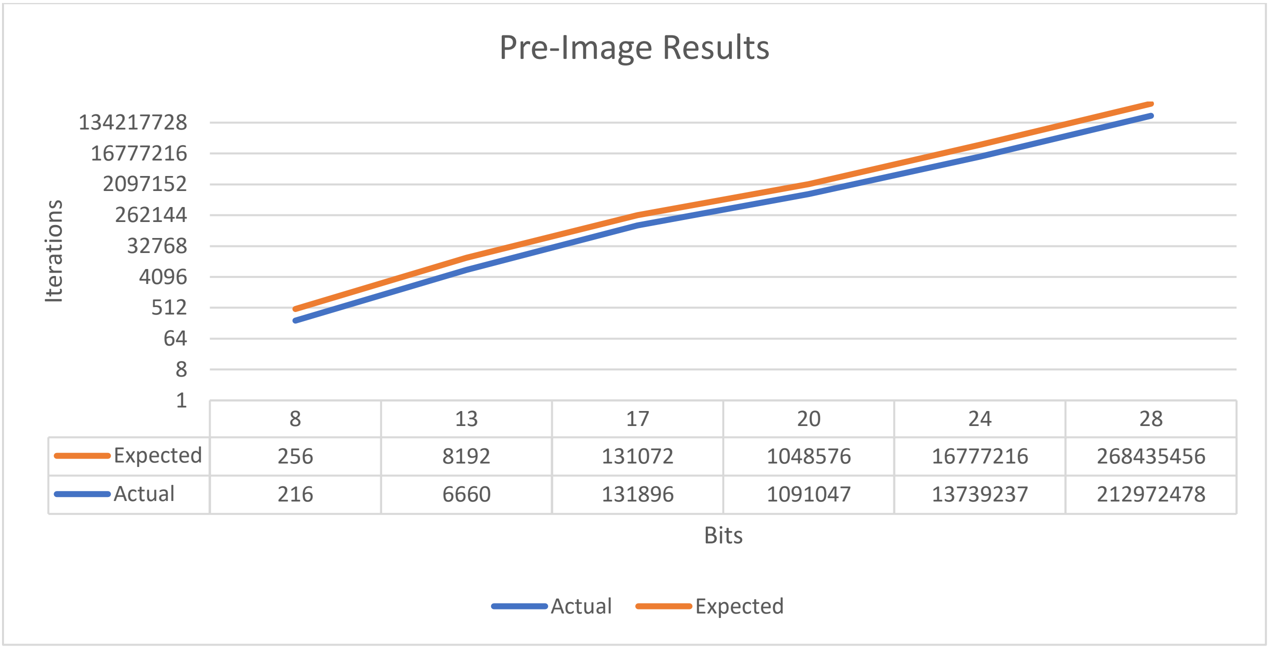
A hash attack is an attempt to find two different inputs that produce the same output when sent through a hash function. If someone can achieve this, they have broken the hash function and can use their faulty "certificate" to gain access to things they're not supposed to. Additionally, they could send data that appears to be authentic because it produces the same hash even though it is faulty.

Accomplishing an attack on any real-world hash function is difficult given the size of the problem. For example, the SHA-1 algorithm produces an output of 160-bits. Sparing the math involved, it would take an inconceivable amount of time to check every possible value. For this report, we will perform attacks on smaller outputs such that a personal computer can find a match in a reasonable amount of time. When we find two unique inputs that produce the same output, we call this a "collision".

Pre-Image Attack

The first type of hash attack we will examine is a **pre-image attack**. This is when we already have a hash output and are trying to find another input that will match it when sent through the hash function.

In this experiment, a random string generator was used to make our initial input. It was passed through a SHA-1 algorithm, and the resulting byte array was cut to the appropriate number of bits we wanted to compare (8, 13, 17, 20, 24, and 28 bits for this experiment). A "while" loop repeated this process, comparing hash outputs over and over until another random string (different from the original) produced the same output. The number of iterations it took to find a collision was then recorded. This process was repeated 100 times for each bit value. The results below reflect the average number of iterations it took for a collision to occur at each respective bit amount. The iterations are rounded up to the nearest whole number.

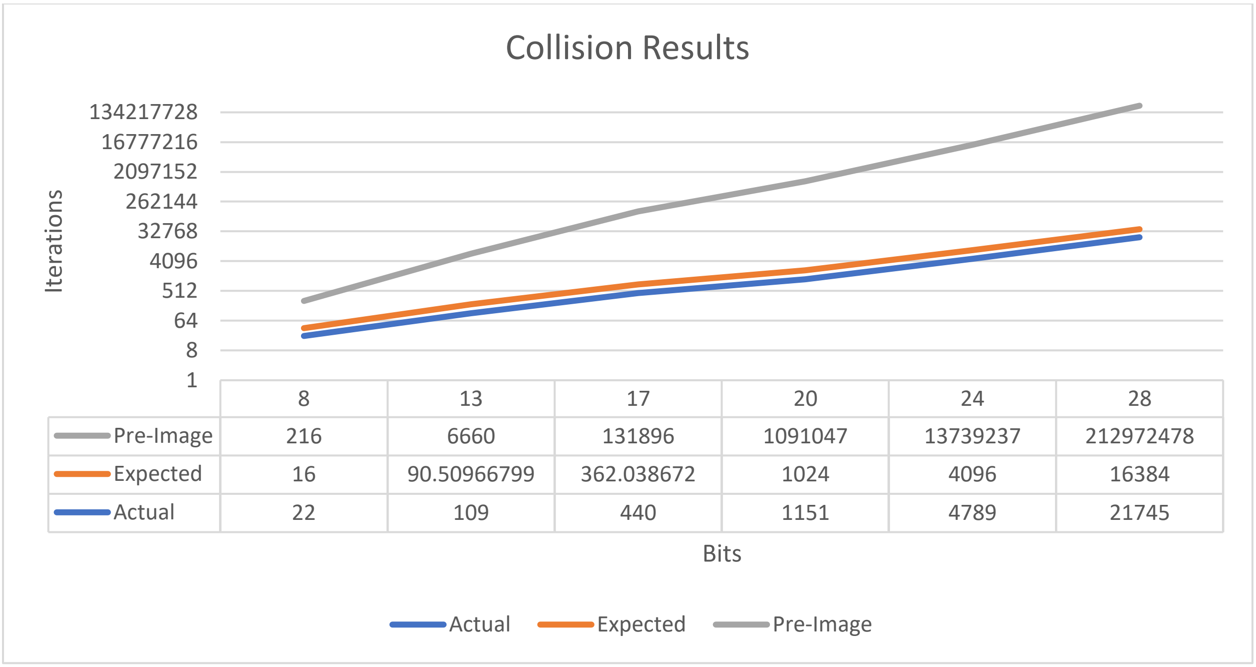


	8	13	17	20	24	28
Expected	256	8192	131072	1048576	16777216	268435456
Actual	216	6660	131896	1091047	13739237	212972478

The expected number of iterations for a pre-image attack is 2^n, n being the number of bits we are comparing. As the graph shows, the actual values versus the expected values followed this pattern closely when compared on a logarithmic scale (base 2). The actual values vary slightly due to outliers and random chance based upon the outputs of the random string generator. To reflect the possible variability, the maximum number of iterations it took to find a collision during the 28-bit trial was 976,389,379, and the minimum number of iterations was 9,258,722. With enough trials, however, the average iterations converged just below the expected amount.

Collision Attack

A **collision attack** is different from a pre-image attack because we are not looking for a random value to produce the same hash output as some original value. This attack produces a bunch of outputs and compares each one to all others before it. We get these unique outputs by starting with a string and adding '1' to the end of it. Each new iteration, the value of the number increments to produce a new string to hash. Because we are looking for a collision amongst any of our iterations instead of trying to find a match against the original, our chances of finding a collision are much higher. This experiment uses the same bit values as the pre-image attack for the sake of comparison, although higher bit values could be used in a reasonable amount of time using this method.



	8	13	17	20	24	28
Pre-Image	216	6660	131896	1091047	13739237	212972478
Expected	16	90.50966799	362.038672	1024	4096	16384
Actual	22	109	440	1151	4789	21745

The expected number of iterations for a collision attack is 2^(n/2). Again, the results follow closely against the expected values when compared on a logarithmic scale (base 2). When compared with the pre-image attack (grey line), we can see how many fewer iterations it takes to find a collision here. Cutting the exponent in half makes a massive difference in iteration count, and therefore a massive difference in the amount of time it takes a computer to find a collision. However, this method does take a considerable amount of storage equal to the number of iterations multiplied by the size of a byte array. The variability between the expected and actual values can be explained the same way as the pre-image attack: outliers and random chance. The highest iteration count for a 28-bit comparison was 46,813, and the minimum was 2,211.

Are All Hash Functions Safe?

Some industry hash functions have been broken and are not safe. MD5 and SHA-0 fall into this category. A weakness in SHA-1 has also been found, so it is not recommended. SHA-2 has no known flaws, but SHA-3 and SHA-256 are recommended as the safest hashing algorithms as of 2022, although SHA-3 doesn't have widespread software and hardware support.

Conclusion

Hashing algorithms provide digital signatures that gatekeep access to information and ensures data maintains its integrity. Hashing algorithms are supposed to produce a unique output for any input to accomplish this. Hash attacks attempt to break this principle by finding two inputs that produce the same output. This is called a collision. A **pre-image attack** is an attempt to find a collision using an original hash and random inputs. This takes an average of 2^n iterations to achieve, n being the number of bits that need to match. A **collision attack** is an attempt to find any two hashes that have the same value. It accomplishes this by comparing each output to all outputs before it. This takes an average of 2^(n/2) iterations to achieve, which is much faster than the pre-image method. People have used these and other methods to break hashing algorithms and exploit the consequences. However, SHA-3 and SHA-256 are considered safe and currently unbreakable.