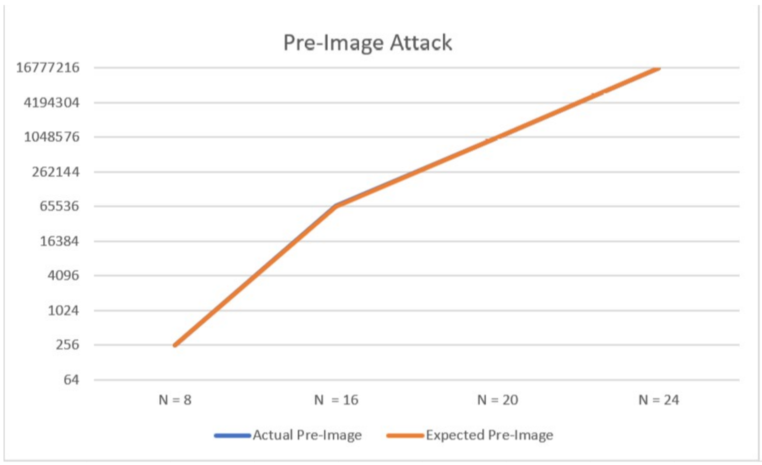
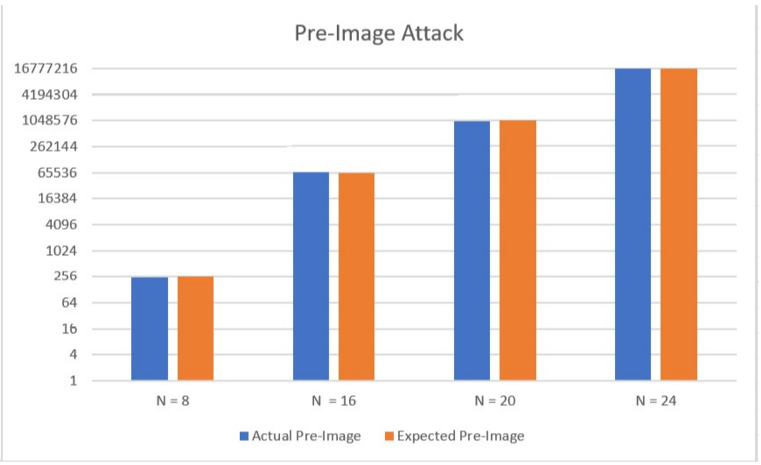
Bryant Hinton

Hash Attack

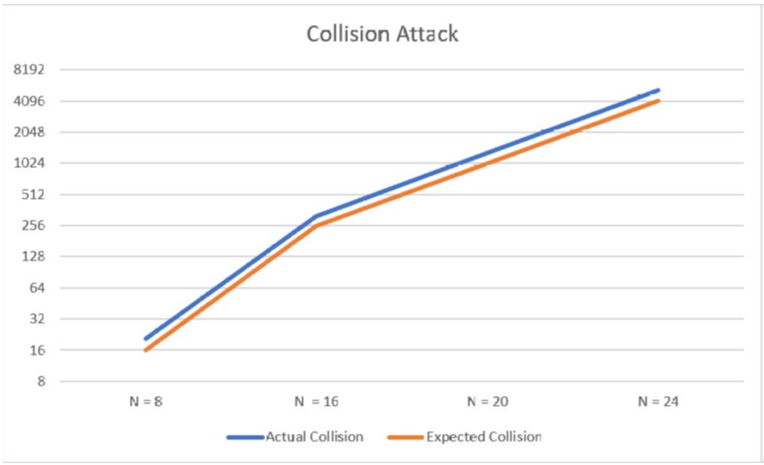
For this experiment on determining the likelihood of success of different types of hash attacks, I had a simple methodology. In python, I used an existing implementation of a hashing library, and used it to create a SHA-1 hash given an input string. Of course, SHA-1 creates a hash 160 bits in length, which would be infeasible to attack in a reasonable amount of time. So, I took an additional argument n, and ran all of my tests with only the n most significant bits of that hash. All of this code was placed in a function and used every time a new hash was needed for testing.

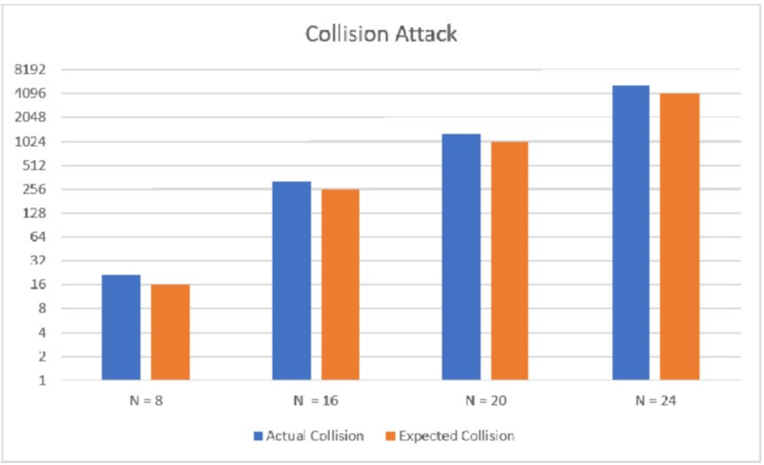
In order to test pre-image attacks, I first use my hashing function to create an initial hash, then I create a random string to be used to generate a second hash. I then compare those two hashes, and if they are unequal, I increment the hex value of my generated string and try again. I do this until both strings match, at which time, I return the number of times I needed to generate a new hash. According to the literature, it should take 2^(n) tries on average to successfully calculate an n-bit hash that equals the initial hash. Each of my own attempts varied wildly, with some attempts taking far fewer tries, and some attempts taking far more. However, with a sufficiently large sample size, I was able to get results nearly identical to those that were expected. As you can see in the “Pre-Image Attack” figure, the numbers are so similar that the lines completely overlap and are indistinguishable. In fact, the expected was a mere 1.0017 times the actual on average for all n that I tested. For each of the four n values, I ran 5000, 1000, 500, and 100 tests respectively.





To compute collisions, I use a very similar approach to that for doing a pre-image attack. The main difference is that instead of calculating an initial hash, and comparing against it every single time, I add that hash to a set data structure, and then simply check if that new hash is already in the set. This has a much lower theoretical time requirement of 2^(n/2) for hashes with a length of n bits. Here my results were quite interesting, with the actual number of tries taking about 25% longer than the theoretical, but with that 25% difference being almost constant along the logarithmic curve. You can see this effect in the "Collision Attack" figure below. I attempted to make the random string used as my nonce longer and shorter to see if length was related, but I received similar results each time. As for why this effect can be seen, I can only surmise that it is because the theoretical number commonly given is simply the big O, and any such constant factors as I saw would be removed when reporting the number. As was the case in the pre-image section, I ran 5000, 1000, 500, and 100 tests for each respective n.





In conclusion, actual experimental results line up very nicely with theoretical predictions in this case. If one were to continue scaling up this experiment to the SHA-1 hash size of 160 bits or higher, one would indeed either have to be extremely lucky, or take an unrealistic amount of time to be able to successfully use these as attack vectors. As least that is the case if only brute force is used as I have done. There may indeed be other weaknesses in SHA that could reduce the search space and allow one of these attacks to be successful, but that is out of the scope of this current experiment.

Reviewed by:

Pearce Solomon