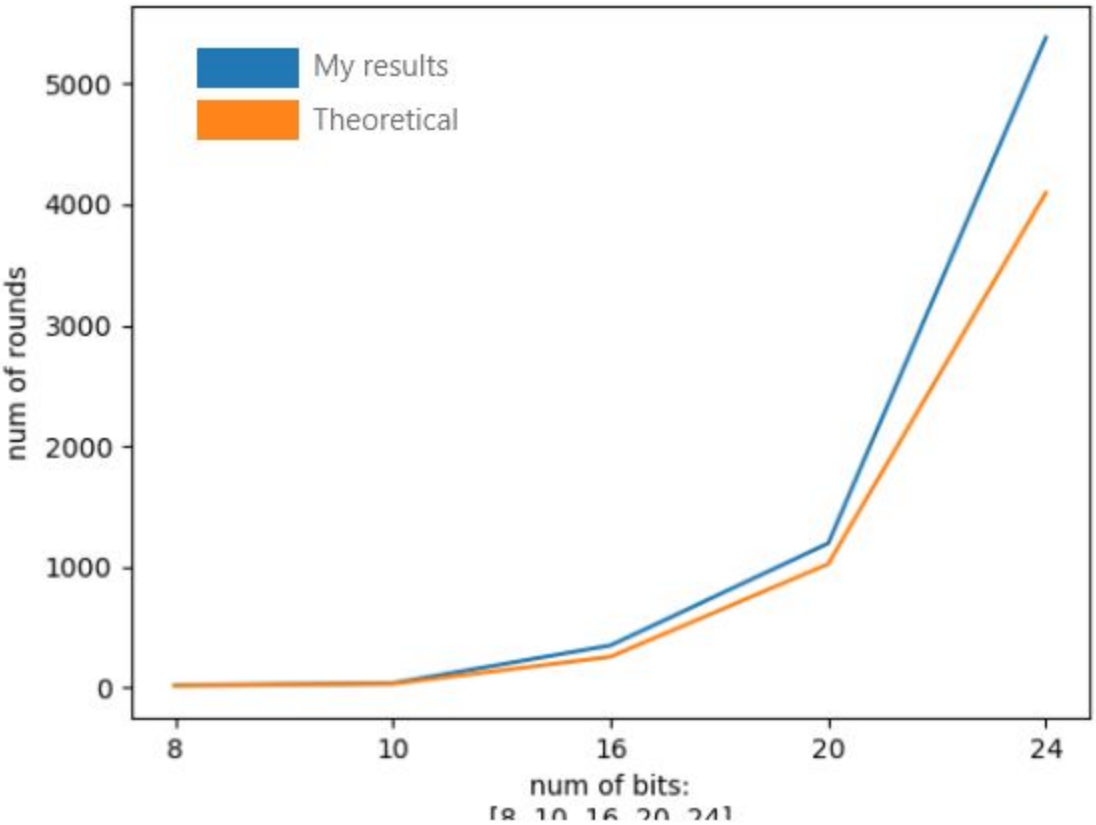
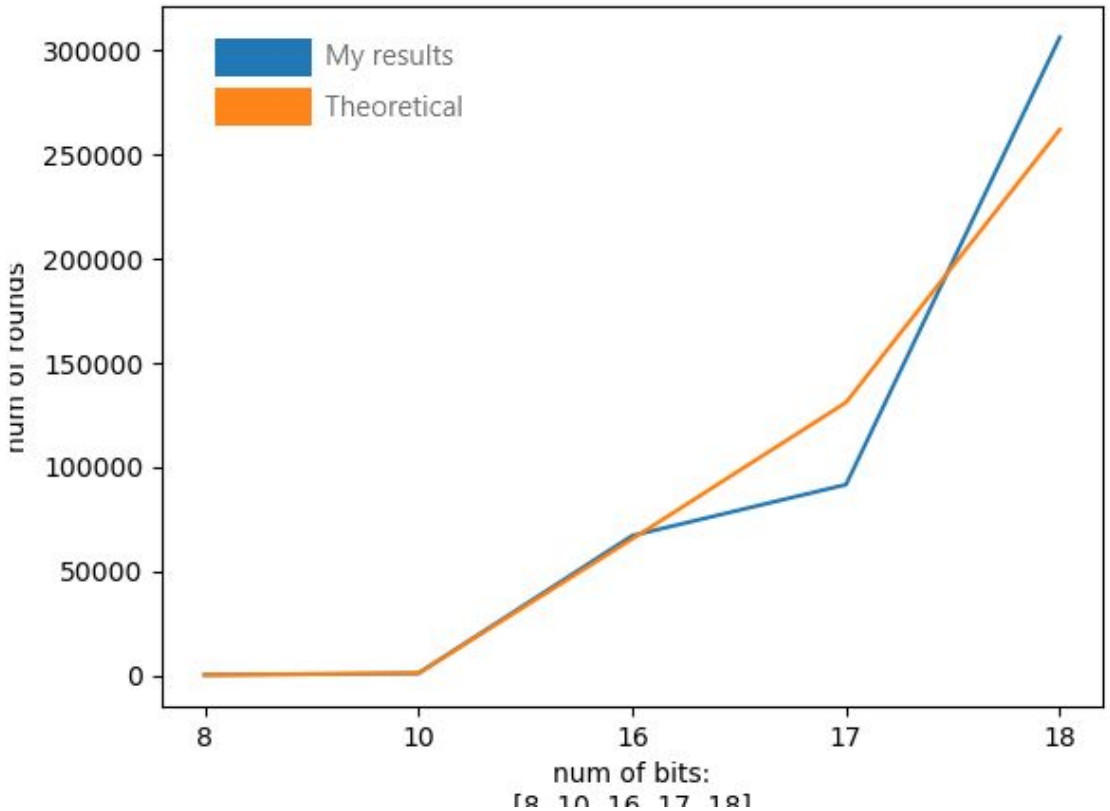
Collision and Preimage Hash Attacks

I went about this experiment trying to reproduce both a collision and a preimage attack. The first and the quicker running attack that I implemented was collision. My goal was to find two strings that would hash to the same digest. In order to carry this out, I created hashes from completely random strings and stored them in a set. I used a set because it is a data structure that keeps track of duplicate insertions. If you insert something into a set that already exists in the data structure, then it will not be put in. Thus, at the beginning of each iteration of hashing and inserting, I check the size of the set, and if it is the same size as it was the last round, then I know that a duplicate was inserted. After testing with 5 different bit sizes (8, 10, 16, 20, 24) and iterating through each size 50 times, I averaged the results and graphed them alongside the theoretical results for a collision attack. The results that I achieved were fairly similar to the theoretical result of 2 ^ (bitSize / 2). I graphed them using a python package called matplotlib:



In order to conduct a preimage attack, a hacker starts out with a specific hash digest. Using that hash as a key, he then uses a hash algorithm (in this case Sha-1) to create hashes from random strings until one of those strings outputs that same hash that matches the key hash. I implemented this by starting my code with a key hash and creating new hashes using random strings in a loop until one of them equaled the key. I then broke out of the loop. Similar to the first experiment, I conducted my preimage attack with 5 different bit sizes and 50 iterations on each size in order to produce more

trustworthy results. However, my code took a long time to run. It never ended! I checked what could help boost productivity and ended up implementing a different way to generate a random string. That sped things up a bit. However, it still was running too long for the 5 bit sizes I had selected earlier, so I decremented the bit sizes quite a bit. After shrinking them, my code finally ran completely through within a couple minutes. Here are the results:



Reviewed by Gabe Suttner