Jesus Garcia
CS 161
23313966
HW4 Writeup

My implementation:

1) Length of Chain and number of chains

c = number of chains e = number of elements in a chain

I created an upper bound on how many chains could be kept in the file with the formula

c = (.93) * (3 * 16 * pow(2, s)) / (round_div(n, BYTE) + 16)

and I created the length of the number of elements in the chain with this formula

e = 2 * (((2^n + (c/2)) / c) + 1)

this basically rounds up the division of, total chains needed divided by the total chains allowed.

2) gentable.c

My implementation of gentable used a counter in the reduce, which was the truncation algorithm. It also used a bitmap of size 2^n of unsigned integers to keep track of the passwords that I had already hashed. This allowed me to skip over those and not create new chains out of them. I implemented a for loop from 0 to 2^n - 1 and used it to write chains to a file.

3) crack.c

In crack I took the fed in hash value, and hashed and reduced that many times with in order to get a password - hash combination where the file's hash value equaled the hash value calculated in crack. Because there were many of these values I test every value and see if somewhere in that chain there existed a password that hashes to the true password's hash.

In order to fully test the matching hash, because I used an extra parameter for my reduce I had to compute the hash chain for the matching password for every value in the range(0, extra_value_size - 1).

4) collisions

In gentable.c the way that I handled collisions was by changing my reduce function. Initially it was just a truncation, but there were many collisions, so ultimately, my reduce function was a truncation and an addition of an extra number to the last byte of the truncation. This extra number was in the range of 0 to 100.

gives a mathematical
relationship between the space used by rainbow (which is proportional to 2 s) and the number of (expected) AES evaluation by crack.

Relationship between AES calls and the space used by the rainbow tables:

My relationship will not be incredibly accurate because of large amounts of AES calls, but for a:

7.2 mb	817,518 AES calls
4.6 mb	59,110 AES calls
1.5 mb	1,390,354 AES calls
731.3 kb	87,377 AES calls

In my implementation I attempted to get my rainbow table to encompass close to every value, however I had a lot of collisions. This may mean that depending on where the extra bit was on calculation of the file's hash-reduce and where the extra bit was in my crack implementation and the number of dead end collisions then the value fluctuated heavily and was not dependent on the size of the rainbow table.

- A 20-bit password with hash 0xae60abdcb19d5f962a891044129d56d4
- 0x31415
- A 24-bit password with hash 0xeb94f00c506705017ce61273667a0952
- 0xfee707
- A 28-bit password with hash 0xa2cf3f9d2e3000c5addea2d613acfda8
- ??????

The ways that I would have tried to optimize my solution if I had time was to lower the max value of my extra number and to write a more unique, better reduce method, rather than just truncation. In crack I would also keep a bitmap of all of the passwords that I had already attempted. Then I wouldn't be doing large amount of redundant work and making many more unnecessary AES calls.

Something that didn't seem to work very well was making my chains longer and containing more passwords. This didn't seem to have a very large effect at all. Something related that worked well though was, after identifying a password that paired to a hashed value that could be obtained from the true passwords hash value, then I would just follow that trail for a longer time over every extra value and this would increase my chances of getting an answer.

Things that worked really well were keeping the already hashed password values in the bitmap and running a for loop over values from 0 to 2^n - 1. This allowed for a wide to complete coverage of the entire password space. Something else that worked really well for me was increasing the value of the extra parameter for truncation to a much larger number.

I think that if I had more time I would attempt to make my gentable do more of the work than my crack file. It seemed like I did a lot of chasing of potential leads in crack that could have been accounted for by having less collisions and also by having more passwords in the chain.