ENEE 457—Project 1
Rainbow Tables Lab

Daniel Xing

October 26, 2018

Abstract

A limited rainbow tables implementation was written in C. `Crack` and `GenTable` are the two main programs that do this, with the main header file `project3.h` containing the majority of the working code. Two types of reduction functions were used. One type is used for plain rainbow tables, and simply takes the first or last n bits to generate a plaintext. The other type used a deterministically seeded PRNG to generate samples of the hash to form the password, and was used for rainbow tables made of rainbow chains. All six challenge hashes were successfully cracked using rainbow chains, and the first four challenge hashes were cracked with plain rainbow tables.

1 Introduction

The project was written in C, and uses the CMake build system to generate makefiles. `GenTable` and `Crack` are the two programs that satisfy the project requirements. `bruteforce` is a program that manually brute-forces a hash's password, and was only used to confirm that the extra challenges were not trick challenges. `tableconvert` can convert ASCII-encoded hex table files into binary table files. ASCII-encoded hex was used for human readability during debugging. `tests` contains toy code snippets for testing ideas.

2 `GenTable`

`GenTable` pseudocode:

```
given a password length n
table = generate_table(n)
export_table(table)
```

GenTable generates a rainbow table for n bits by calling `generate_table()`. The number of AES executions is then pulled from the hash function and printed to `stdout`. The rainbow table is then saved to disk in binary format using `export_table()`.

2.1 generate_table

generate_table pseudocode:

```
while there's still table entries to generate
    generate_chain
append chain head and tail to table
```

generate_table takes a pointer to a table struct, allocates the appropriate amount of memory to store all heads and tails of the chains, and starts calling generate_chain in a while loop. A progress meter periodically prints the percent of table entries that have been generates. hashcount is only used to get the status of the generate_chain function.

2.1.1 generate_chain

generate_chain pseudocode:

```
pick a random plaintext password
while we haven't finished the chain
    hash the current plaintext
    reduce the current hash
return the head and tail of the chain
```

generate_chain takes n and a pointer to a table entry. It generates a random plaintext by calling generate_random_plaintext, then in a while loop repeatedly hashes and reduces the plaintext. Preprocessor statements here enable/disable checking for whether or not a specific hash has been generated yet, enabling/disabling rainbow chains, and ignoring or removing duplicate tails in the table.

3 Crack

Crack pseudocode:

```
given a password length n and hash
import the rainbow table
search_table
if we don't find the password
    report failure
else
    report success, password
```

Crack takes in n and a hash. It first imports a rainbow table by calling import_table. Then it searches the table by calling search_table. If search_table finds a password, it prints the password and the number of AES encryptions performed before exiting. Otherwise, it will report failure and print the number of AES encryptions performed before exiting.

3.1 search_table

search_table pseudocode:

```
for i = 1 to the entire length of a chain
    if using plain chains
        reduce
    else if using rainbow chains
        reduce and hash an appropriate amount of times for a rainbow chain
    search the table tails
    if we found a match
        search the chain
        if we found the password in the chain
            return the password
    if using plain chains
        hash
return failure
```

search_table takes a hash and searches the table. If it finds a matching password, it saves it to plaintext. In a for loop that executes $2^{\frac{n}{2}}$ times, it will repeatedly reduce and hash, the exact way it does so depending on whether or not rainbow chains are being used. It will then search the rainbow table for the plaintext, and if it finds a matching plaintext at a table tail, it will then search the chain using search_chain. Because of the nature of rainbow chains, where the order of the applied reduction functions matters, a for loop is used to apply the reductions in the correct order.

3.1.1 search_chain

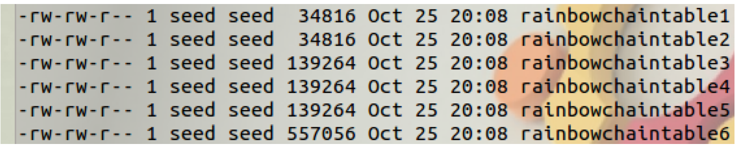
search_chain pseudocode:

```
for each hash in the chain
    if the hash in the chain matches the hash we're trying to crack
        return the corresponding plaintext
    reduce
    hash
return failure
```

search_chain just repeatedly calls hash and reduce, checking hashes against the target hash until we find a match, very similar to generate_chain.

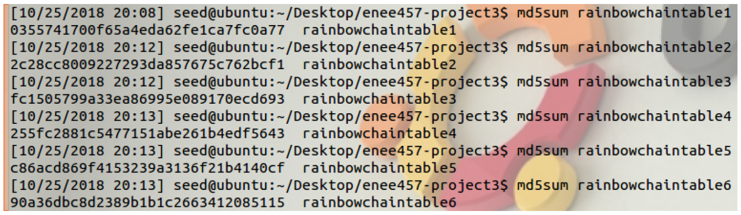
4 Rainbow Table Sizes

Screenshots for the six different rainbow chain table sizes are in figure 1. A screenshow containing the MD5 hashes of the six rainbow tables are in figure 2.



```
-rw-rw-r-- 1 seed seed  34816 Oct 25 20:08 rainbowchaintable1
-rw-rw-r-- 1 seed seed  34816 Oct 25 20:08 rainbowchaintable2
-rw-rw-r-- 1 seed seed 139264 Oct 25 20:08 rainbowchaintable3
-rw-rw-r-- 1 seed seed 139264 Oct 25 20:08 rainbowchaintable4
-rw-rw-r-- 1 seed seed 139264 Oct 25 20:08 rainbowchaintable5
-rw-rw-r-- 1 seed seed 557056 Oct 25 20:08 rainbowchaintable6
```

Figure 1: Screenshot of file sizes.



```
[10/25/2018 20:08] seed@ubuntu:~/Desktop/enee457-project3$ md5sum rainbowchaintable1
0355741700f65a4eda62fe1ca7fc0a77  rainbowchaintable1
[10/25/2018 20:12] seed@ubuntu:~/Desktop/enee457-project3$ md5sum rainbowchaintable2
2c28cc8009227293da857675c762bcf1  rainbowchaintable2
[10/25/2018 20:12] seed@ubuntu:~/Desktop/enee457-project3$ md5sum rainbowchaintable3
fc1505799a33ea86995e089170ecd693  rainbowchaintable3
[10/25/2018 20:13] seed@ubuntu:~/Desktop/enee457-project3$ md5sum rainbowchaintable4
255fc2881c5477151abe261b4edf5643  rainbowchaintable4
[10/25/2018 20:13] seed@ubuntu:~/Desktop/enee457-project3$ md5sum rainbowchaintable5
c86acd869f4153239a3136f21b4140cf  rainbowchaintable5
[10/25/2018 20:13] seed@ubuntu:~/Desktop/enee457-project3$ md5sum rainbowchaintable6
90a36dbc8d2389b1b1c2663412085115  rainbowchaintable6
```

Figure 2: Screenshot of md5 hashes of the rainbow tables.

5 Reduction Functions

Two general types of reduction function are used: simply taking the first or last n bits of the hash, and using a PRNG to pick out random set of $n/4$ 4-bit blocks. The first type is found in the `reduce` function, and the second type is found in `new_reduce`.

`reduce` simply takes the last (or first, depending on preprocessor defines) $n/4$ 4-bit blocks and uses them to form a new valid n bit plaintext with appended zeros. It does this in a for loop. Great care was taken in cases where n mod 4 = 1.

`new_reduce` takes advantage of C's `rand` and `srand` functions. `rand` is notorious for being a very predictable PRNG, especially if you do not seed it with `srand`, but we use that to our advantage here. `srand` is seeded with the index in the chain to produce a unique set of hash indicies that we will extract from to form the plaintext. Due to the very large cycle of `rand`, for every chain index we are guaranteed a different set of offsets, and therefor a different reduction function for every position in the rainbow chain.

6 Results

Screenshots are attached.

Found hashes:
000000000000000000000000000BF6F1 (password, 20 bit)
8DE0BCFFE587F63ED5C823DCF9BF5131 (hash)

0000000000000000000000000003F4B0 (password, 20 bit)
F7EF413CC51DF04ABF6872DB315E694B (hash)

00000000000000000000000000008FD2EE (password, 24 bit)
ED078D9B527A81FE4725228D88B664AE (hash)

000000000000000000000000004BE9CD (password, 24 bit)
AE955B027A3D0CB5401B63B4D26A10BA (hash)

00000000000000000000000000A492F2 (password, 24 bit)
B8A1C2B0AFFBF389D6F0FC0584CCEFB2 (hash)

00000000000000000000000009BF210D (password, 28 bit)
86527077E1CB39B6B2E6F414B1A758F6 (hash)