Fernando Blanco Albendea

Project 3: Rainbow Tables

Report:

I have used #C language for this project:

Gentable Pseudocode: This program computes $2^{n/2}$ chains given the number n. Each chain is computed generating a random password, i.e., an array of 128 bits (or 16 characters, if each character has a length of 1 Byte) where the first 128-n bits are 0, and the last n bits are random. From this password, the chain consists of $2^{n/2}$ passwords, and $2^{n/2}$ hashes, alternating. Being 'r' a reduction function and 'h(x)' a hash function of an all-zero plaintext using the password x, the rainbow table will look like this:

p-h(p)-r(h(p))-h(r(h(p)))-r(h(h(h(h))))-.....

However, the program will only print the first and the last point of each chain. Each start point and each endpoint will be printed in the rainbow table file.

```
Initialize variables;
Plaintext is all-0;
Store n and the seed to generate the random numbers from the arguments line (argv);
Open the rainbow table file that will be used to store the start and the end points;
for(int i = 0; i<2^(n/2); i++){

        //Generating a random password
        for(int k = 0; k<128-n; k++)
                password[k]=0;
        for(int k = 128-n; k<128; k++)
                password[k] = random number ∈(0-9)U( A-F);

        Print the password (start point) in the rainbow file;
        for(int j = 0; j<2^(n/2); j++){
                ciphertext = Enc_password(plaintext);
                ciphertext = reduction_function(ciphertext);
        }

        Print the ciphertext (end point) in the rainbow file;
}
Close the rainbow table file;
```

Crack pseudocode: This program takes the given number n and the given hash h. Then, it reads the endpoints from the rainbow table file (the even lines) one by one, and for each endpoint, it computes the hash h(endpoint) and compares it with the given hash. If they are equal, the password is the endpoint. If they are not equal, it reduces and hashes the file (F(h) = h(R(h))), and then it compares this with all the endpoints again. This iteration is repeated until a password is found, or $2^{n/2}$ times.

```
Initialize variables;
//k is the point in each chain I am in
//m is the chain number
Variables m = 0, k = 0;
Plaintext is all-0;
Store n and h(p), obtained from the arguments line (argv);
Open the rainbow table file that will be used to read the start and the end points;
Copy h(p) in h_Copy(p);
while(chain not found && k<2^(n/2)){
        m = 0;
        while(chain not found && m<2^(n/2)){
            Skip the next line in the rainbow file;
            Read line(endpoint);
            h(endpoint) = Enc_endpoint(plaintext);
            if(h(endpoint) == h_Copy) {
                PASSWORD FOUND!!;
                Final_password = endpoint;
            }
            m++;
        }
        if(password not found){
            h_Copy(p) = reduction_function(h_Copy(p));
            New_Hash = Enc_h_Copy(p)(plaintext);
            h_Copy(p) = New_Hash;
            k++;
}

if(PASSWORD FOUND){
        printf("The password is :%s", finalPassword);
}
else{
        printf("No password found");
}
```

This is an example of a rainbow table generation when n = 16. I have decided to write each start point and end point in a different line. This is how the file looks like. It can be seen that the size is in the required boundaries.

rainbow.txt

```
00000000000000000000000000004a59
00000000000000000000000000000d625
00000000000000000000000000005741
00000000000000000000000000000044f
00000000000000000000000000005146
00000000000000000000000000000f289
00000000000000000000000000004446
00000000000000000000000000000020
00000000000000000000000000004157
00000000000000000000000000009615
00000000000000000000000000000434d
00000000000000000000000000022f3
00000000000000000000000000005551
00000000000000000000000000000c5b0
00000000000000000000000000004753
00000000000000000000000000005291
00000000000000000000000000004448
00000000000000000000000000001d75
00000000000000000000000000004841
00000000000000000000000000007740
00000000000000000000000000005854
00000000000000000000000000003ede
00000000000000000000000000004549
00000000000000000000000000011fa
00000000000000000000000000005348
00000000000000000000000000000f10e
00000000000000000000000000004358
00000000000000000000000000000bf34
00000000000000000000000000000554b
00000000000000000000000000012dd
00000000000000000000000000004646
00000000000000000000000000000a24b
00000000000000000000000000004c42
00000000000000000000000000000f8e5
00000000000000000000000000004644
00000000000000000000000000030c8
00000000000000000000000000000494b
00000000000000000000000000000ac15
00000000000000000000000000004a4a
00000000000000000000000000001a62
00000000000000000000000000000474e
00000000000000000000000000000787e
00000000000000000000000000005841
00000000000000000000000000002feb
```

```
[10/24/2018 21:46] seed@ubuntu:~/Project$ ./Gen 16 3434
256
[10/24/2018 21:47] seed@ubuntu:~/Project$ md5sum rainbow.txt
220c126184c59243775a20b0222efb05  rainbow.txt
[10/24/2018 21:47] seed@ubuntu:~/Project$ ls -l
total 3568
-rwxrwxr-x 1 seed seed 1200215 Oct 24 21:46 Crack
-rw-rw-r-- 1 seed seed    2219 Oct 24 21:34 Crack.c
-rwxrwxr-x 1 seed seed 1200231 Oct 24 13:34 enc
-rwxrwxr-x 1 seed seed 1200231 Oct 24 21:46 Gen
-rw-rw-r-- 1 seed seed    2041 Oct 24 13:32 GenTable.c
-rw-rw-r-- 1 seed seed     166 Oct 24 13:35 Makefile
-rw-rw-r-- 1 seed seed     112 Oct 22 16:15 Makefile~
-rw-rw-r-- 1 seed seed   16896 Oct 24 21:47 rainbow.txt
-rw-rw-r-- 1 seed seed      17 Oct 23 16:57 rainbow.txt~
[10/24/2018 21:47] seed@ubuntu:~/Project$
```

Fernando Blanco Albendea

The reduction functions I have tried do not work, but the idea would be to try to take the last n bits (or n/4 characters) from the hash we want to reduce (hash[j]) and try one of the following options:

1. hash[j] = $2^{n/2}$ mod hash[j]
2. hash[j] = $2^{n/2}$ mod (hash[j]/(j+1))
3. char a = hash[0]+hash[1]+…+hash[j]
   hash[j] = a mod $2^{n/2}$
4. char a = hash[0]+hash[1]+…+hash[j]
   hash[j] = a mod (j+1)
5. hash[j] = hash[j] mod ((hash[j]/(j+1)) mod $2^{n/2}$)

These are some options of all the reduction functions that I have been trying.