**CS 161 – Computer Security**
Instructor: Tygar
7 October 2014

# Homework 4

## Notes

- Homework 4 is due on ~~14 October~~ extended to 21 October 2014 at 3PM.
- Please work on this homework individually – no collaboration allowed.
- **Please list your name, student ID, section, and TA at the top of your solution**
- To submit your homework: please create a `~/hw4` directory in your class account. Create a tarball (including a Makefile and source code) entitled `hw4.tar`. Put your writeup in `hw4.pdf`. Run the command "`submit hw4`" from inside the `~/hw4` directory

## Assignment

In this assignment you will create a rainbow table and use it to "break" hash functions. You will write a program to generate a rainbow, a program to invert hashes using the rainbow table, you will prepare a write-up describing your strategy and you will invert three hashes.

We will use the following system to hash $n$-bit password: We will left-pad the password with 0s until it is 128-bits long, calling the result $P$. Then we will compute AES-128 using $P$ as a key on plaintext block of all zeros

$$H(P) = AES_P(0)$$

Thus, for the 12-bit password $P$ =`0xABC`, the result should be

$$H(P) = \texttt{0x970fc16e71b75463abafb3f8be939d1c}$$

In this assignment, you may assume that $n$ is less than 32. You are given $H(P)$ and $n$ and must recover $P$. One way to do this would be to perform a brute-force attack using on the order of $2^n$ AES evaluations. Alternatively, you can pre-compute all $2^n$ possible hashes and then find $P$ in nearly constant time; this requires $O(2^n)$ space. The goal of using a rainbow table is to do better. Success in this assignment requires finding an implementation that uses significantly less than $2^n$ time and space.

Both programs should be written in C (and should include an appropriate Makefile) and should run on the hive departmental machines. Use this implementation of AES: https://polarssl.org/aes-source-code

The first program is `gentable`. It takes two command-line arguments, and outputs to a file `rainbow`. The first command-line argument is the password length $n$ measured in bits. The second argument $s$ determines a bound on the size of `rainbow`; it must be no larger than $3 \times 128 \times 2^s$ bits (or $3 \times 16 \times 2^s$ bytes). (If you pre-computed all hashes, then $s = n$). You may assume that $n - s \leq 10$. You must meet this space limit to earn credit for the problem. You can use `ls -l` to check the size of your file.

The second program is `crack`. It takes three command-line arguments, and outputs to `stdout`. The first two command-line arguments are the same as for `gentable` (again, you may assume that $n - s \leq 10$). The final argument is $H(P)$ in hex. When you run `crack` $n$ $s$ $H(P)$, you may assume that `gentable` $n$ $s$ was previously run to generate `rainbow`. The output of `crack` includes two items: the password $P$ or "failure," and the number of times AES was evaluated. Make sure your program accurately reports the number of AES evaluations; if this is not correct, you will not receive credit for the assignment. Thus running

```
% gentable 12 12
% crack 12 12 0x970fc16e71b75463abafb3f8be939d1c
```

will give output

```
Password is 0xABC.  AES was evaluated 191 times
```

assuming that in this execution of `crack`, AES was actually evaluated 191 times.

You should include a 1-2 page writeup that describes your implementation, and 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`. Your writeup should include a discussion of how you addressed the problem of collisions in your rainbow table chains. Finally, your writeup should include the inversions of the following three password challenges:

- A 20-bit password with hash `0xae60abdcb19d5f962a891044129d56d4`
- A 24-bit password with hash `0xeb94f00c506705017ce61273667a0952`
- A 28-bit password with hash `0xa2cf3f9d2e3000c5addea2d613acfda8`