Homework – Rainbow Tables
Due at 11:59PM on Nov. 28

As discussed in class, assume the following scheme is being used to hash passwords: An $n$-bit password $P$ is padded to the left with $128 - n$ zeros and used as an AES-128 key to encrypt the all-0 plaintext; the result is the “hashed password”. I.e.,

$$H(P) \overset{\text{def}}{=} \text{AES}_{0^{128-|P|} \parallel 0^{128}}(P).$$

So for the 12-bit password $P = 0\text{ABC}$, the result should be

$$H(P) = 970fc16e71b75463abafb3f8be939d1c.$$

You may assume $n$ is a multiple of 4.

The scenario is that you are given $H(P)$ and $n$ and need to recover $P$. This can always be done by a brute-force attack using about $2^n$ AES evaluations. Alternately, one can pre-compute all $2^n$ possible hashes and then find $P$ in essentially constant time; this requires $O(2^n)$ space. The goal of using a rainbow table is to do better. I don’t care much about the low-level details of how you implement the attack, as long as your attack uses significantly less than $2^n$ time and space.

1. Write two programs, called GenTable and crack. The first of these corresponds to the pre-processing phase in which you generate a rainbow table, while the second corresponds to the on-line phase in which you are given $H(P)$ and need to recover $P$.

   - **GenTable** should take two command-line arguments, and generate output to a file rainbow. The first command-line argument will be $n$, the password length (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. (So the trivial attack in which you pre-compute hashes of all passwords would have $s = n$. The factor of 3 in front can be lowered, but I include it to allow you a little “slop.”) Failure to meet this space bound will result in 0 points. You can use `ls -l` to check the ASCII character length of your file.

   - **crack** should take three command-line arguments, and generate output to stdout. The first two command-line arguments are exactly as above. The final argument is $H(P)$ in hex. When you run `crack n s H(P)`, you may assume that GenTable $n$ $s$ was just run to give rainbow.

The output of crack should include two items: the password $P$ or “failure”, and the number of times AES was evaluated. (See next.) Failure to report the number of AES evaluations accurately will be considered cheating, and will result in 0 points.
• So running
  % GenTable 12 12
  % crack 12 12 970fc16e71b75463abaf3f8be939d1c

  should give output “password is ABC, AES evaluated 191 times” (assuming that
  in this execution of crack AES was evaluated 191 times).

2. Include a 1–2-page writeup that describes your implementation, and gives a math-
  ematical relationship between the space used by rainbow (which is proportional to 2^s)
  and the number of (expected) AES evaluations by crack.

3. Use your programs to recover the passwords from the two challenges here:

  http://www.cs.umd.edu/class/fall2012/cmsc498L/rainbow.txt

  Include the answers at the end of your writeup.

Submit your homework by emailing jkatz@cs.umd.edu a tarball containing: (1) source
  code for your programs; (2) a Makefile; (3) your writeup.