**sdafCSE 410: Introduction to Computer Security Fall 2016**

**Department of Computer Science and Engineering University at Buffalo**

**10/05/2016 Due: Tue, Midnight**

**Homework #4**

General instructions:

* You are allowed one submission attempt on UBLearns.
* If in doubt, use all available resources to ASK QUESTIONS (piazza, office hours, etc.)
* Show all steps, and cite all resources used for answering the questions
* No late submissions are accepted.
* Deadline is in the header of the assignment.
* Academic integrity is taken seriously, and cheating is not tolerated.
* Individual homework – don’t collaborate with others on this homework.

**Notes and homework instructions:**

* This homework consists of only 1 question, and all parts are mandatory except (1.b).
* Parts 1-4 require coding (total of 6 functions).
* Coding must be done in Python.
* Your code should be a single .py file, with as many functions as requested.
* Each function should be named with the corresponding question (e.g., Q1A, Q1B, etc.).
* The output of each function should be in a file named as the function (e.g. Q1A.txt)
* A long with your code, provide a short report that outlines steps followed and comments (where requested) to answer the questions below.
* In your code provide a main function that would iterate over all functions and generate the requested files (see above).
* Provide your submission as a single compressed file (.zip) named [your-ubitname].zip
* The code will be automatically graded, so provide what is requested exactly.
* If in doubt, use available resources (piazza, TA, OH, lecture) to get your doubts cleared.

Note about data: do not distribute the passwords file. When unzipped, the file is about 2.8GB

Question 1 (100pts) attached you will find a large file with a large number of repeating ASCII passwords (320,412,510 passwords) corresponding to 2,151,219 unique passwords. Based on the contents of the file and what you learned in the class, perform the following (using python):

1. **Frequency analysis:** 
   1. **(Coding)** Perform a frequency analysis of the passwords in the attached file. The outcome of this analysis should be a SORTED FILE with three columns (sorted based on the first column). The first column should be the passwords. The second column should be the number of times that each password in the first column has appeared in the passwords file. The third column should be the probability of the password in the file (this is, the third column will be the value in the second column divided by the number of passwords in the given file; 320,412,510). Columns in the output file should be separated by a tab (this is, “\t”). The last column should be rounded to 10 decimal digits.
   2. **(Bonus, coding) Dictionary comparison:** A character-level frequency of alphabets in the English language based on 40,000 words is shown in Appendix 1. Generate the equivalent table from the passwords dictionary you have created in question 1. Discard special characters in calculating the frequency and probability of characters. Your file should be 3 tab-separated columns, where the first column is a letter, the second one is the number of times it appeared in all passwords, and the last is the probability distribution of count for the character (as a percent). Use 2 decimal digits as in the appendix. Consider all passwords uppercase for this question only.
2. **Strength analysis:** 
   1. **(Coding)** Calculate the strength of dictionary you calculated in 1.a above using the Shannon entropy (called the effective entropy). The result should be a single positive number, rounded to 3 decimal digits.
   2. **(Coding)** Calculate the ideal maximal entropy achievable for this dictionary, and compare it to the effective entropy.
   3. Comment on why this maximal entropy not achieved.
3. **Strength analysis of unique passwords:** 
   1. **(Coding)** Redo question 2, but this time using the set of unique passwords in column 1 of question 1.a. This is, when calculating the probability as in question 1.a, consider your space of events to be driven from the unique passwords dictionary, rather than the password file.
   2. Compare the results with the answer of question 2.
4. **Offline dictionary attacks: (Coding)** The hash values (MD5 hashes) below were revealed when a login database was compromised. Find the passwords of the following hashes, given that they are in the dictionary above.
   1. ba931c15ec0163c4bb339f41571694ce
   2. c9cd905fc459e5550b8c3b01d4346c25
   3. e9269d9e52a692f52caece9d0e7cdae1
   4. 660719b4a7591769583a7c8d20c6dfa4

Your code should be as efficient as possible (computationally). This is, for a given hash value corresponding to a password that is at the i-th row in the sorted file, your code should run for at most i-iterations (i.e., calculation of a hash value, comparisons, etc.)

The output of your code should be a 3 columns tab-separated file, with the first column being the md5 above, the second column being the corresponding password, and the last column being the rank (index) of the password in the sorted dictionary. The rankings start with 1.

1. **Contexts for targeted password cracking:** Jim is a fan of the Buffalo Bills. This, in turn, is reflected in his choice of passwords. Often time, he would use his passwords as words related to the Buffalo Bills, including names of players, years of wins, etc. The MD5 (unsalted) hash value of Jim’s password is “83bfc234f88cc75d52e9ec24e54bc8be”. Answer the following:
   1. Provide 3 possible candidates for what the password could be
   2. If you were told that the password is a family name of a player playing for the Bills, how would this help you find the password? How would this reduce the password space?
   3. If you are told that the password is a family name of a player who retired on April 7th of 2016, how would this help you find the password?
   4. Use a side channel to infer the password, without knowing the context above (e.g., search the MD5 in the Google search engine). Provide any potential results. Confirm whether your search result is correct or not.

**Appendix: character frequency based on 40,000 words in English.**

|  |  |  |  |
| --- | --- | --- | --- |
| Letter | Count |  | Frequency |
| E | 21912 |  | 12.02 |
| T | 16587 |  | 9.10 |
| A | 14810 |  | 8.12 |
| O | 14003 |  | 7.68 |
| I | 13318 |  | 7.31 |
| N | 12666 |  | 6.95 |
| S | 11450 |  | 6.28 |
| R | 10977 |  | 6.02 |
| H | 10795 |  | 5.92 |
| D | 7874 |  | 4.32 |
| L | 7253 |  | 3.98 |
| U | 5246 |  | 2.88 |
| C | 4943 |  | 2.71 |
| M | 4761 |  | 2.61 |
| F | 4200 |  | 2.30 |
| Y | 3853 |  | 2.11 |
| W | 3819 |  | 2.09 |
| G | 3693 |  | 2.03 |
| P | 3316 |  | 1.82 |
| B | 2715 |  | 1.49 |
| V | 2019 |  | 1.11 |
| K | 1257 |  | 0.69 |
| X | 315 |  | 0.17 |
| Q | 205 |  | 0.11 |
| J | 188 |  | 0.10 |
| Z | 128 |  | 0.07 |