[TODO: think of a title for this. be creative? lol…we’ll see.]

Xixia Wang

Junior Paper: Computer Science

Dr. Joseph Bonneau

April 29, 2014

Abstract

Passwords are everywhere. In modern society, the digital world and thus the web has become an ingrained part of our lives, with passwords being typically the only layer of security protecting our online information. Due to the ubiquity of passwords for online security, it is vital to study passwords and understand how easily they can be cracked by other, typically malicious, sources.

There already exists a number of studies done on Western-language passwords from leaked websites, which examine the efficacy of different cracking methods, such as through the dictionary attack, probabilistic context free grammar models, or Markov models. Yet, because the focus is on Western languages, usually English (as the websites of the leaked passwords have a user base of mainly English speaking individuals), there is little known about how well these password cracking methods work on other languages. Thus, this study will specifically examine Chinese password sets and utilize the probabilistic context free grammars (PCFG) model in order to determine a more effective way to crack the passwords, compared to a typical password cracker (John the Ripper). (TODO: clean this up…and can definitely add more substance).

[TODO: DA TITLE. PROB IMPORTANT. MUST DO…]

**1. Introduction**

“You have a secret that can ruin your life,” is the opening line in a 2012 *Wired* article, highlighting the security vulnerabilities that underlie passwords, and making a call to “Kill the Password.” [1] Indeed, it seems shocking, to a degree, that a string of characters is the primary safe keeper of our personal information online, yet this occurrence is unlikely to change in the near future. Thus, there is little doubt in the importance of passwords in our lives today. The majority of actions we perform on our computers involve some sort of self-authentication, especially in the use-cases of trying to access someone’s data, usually our own. Due to the prevalence of passwords, it is increasingly important to maintain a secure password, which means one that is not easily guessed by an adversary and his algorithm. Thus, there have been numerous studies done on the strength of a password, and subsequently, just as many papers published on the topic. In general, when it comes to offline cracking, two methods prevail: using probabilistic context-free grammars (PCFG for short) or using Markov models (TODO: double check this? cite it?). This has been discerned from numerous studies and tests involving leaked password databases, most notably the Rockyou dataset (TODO: cite?).

* 1. **The Problem**

The number of studies and research done on leaked password datasets from primarily English websites is indeed robust, and has led to a clearer understanding of what makes a password more secure as well as a better measurement standard for the security of a password. However, not much analysis has been done thus far on Chinese language passwords, even though there are over 1.2 billion speakers in the world. In fact, due to the high population, it is of little surprise that China has nearly twice as many Internet users as the United States has people [2], numbering up to 618 million Internet users.

Of course, it might be worth asking why it is of importance to study passwords from primarily Chinese sources, and the reasons can be broken down into two primary categories: for the purposes of examining the strengths of our current password cracking methods and to further promote the security of online users. With more and more Chinese users on the Internet as well as mobile devices, there is subsequently more and more incentive for both companies and adversaries to target Chinese audiences. Due to the sheer quantity of Chinese users online, it would be a good opportunity to explore the efficacy of certain password cracking techniques applied to Chinese-language datasets, in order to determine their effectiveness in general.

**2. Chinese Passwords: an Overview**

First and foremost, it is vital to understand the fundamental differences in the composition of passwords between datasets from English websites against Chinese websites. Much of our understanding of Chinese-language passwords is derived from the paper “Of contraseñas, סיסמאות, and 密码: Character encoding issues for web passwords” (TODO: figure out how to get this to show up properly ><) [3], by Bonneau and Xu. Within it, they examined two leaked Chinese-language datasets: 70yx and csdn. A brief overview of the sites is given below[[1]](#footnote-1).

1. 70yx.com: 70yx is primarily an online gaming website, where one can download online multiplayer role-playing games. This dataset consists of approximately 10 million usernames and passwords.
2. csdn.net: CSDN, which stands for “Chinese Software Developer Network”, provides forums and blog hosting, and is considered one of the largest networks of software developers in China. This dataset numbers around 6 million passwords.

It stands to reason that because of the language differences, and because Chinese is based on a character system, then Chinese-language passwords will be much more difficult to brute force due to the fact that it has a larger character set to choose from. However, this is surprisingly not the case in the majority of the leaked Chinese-language password datasets that were studied; in fact, most of the passwords primarily used ASCII characters. Specifically, the 70yx dataset had a rate of 99.3% of all passwords were completely alphanumeric, and the entire dataset was at least partly alphanumeric. The CSDN dataset boasted similar percentages, where 98.3% of all passwords were completely alphanumeric, and, again, 100% of them were at least partly alphanumeric. What this means is that even though the Chinese language stems from a character system, which consists of several thousands of characters, the vast majority of passwords are still written in alphanumeric characters.

Another noteworthy aspect of Chinese-language passwords is their reliance on digits. In the 70yx and CSDN datasets, approximately 48.1% and 45.0%, respectively, of all passwords consisted of *only* digits (0-9). A little less than half of all passwords in both datasets contained only numbers, which indicates a fundamental difference in password creation between English-language passwords and Chinese-language. When looking at passwords that contain some digits in the two datasets, that percentage rises to 90.8% and 87.1%, respectively. This strong reliance on digits in passwords is not seen in English-language datasets, such as Rockyou, where only 15.9% of the passwords contain *only* digits, and 54.0% of the passwords contain some digits. There is no definitive reason behind this, but one can speculate that the higher inclusion of digits is related to how many Chinese speakers may not be accustomed to using the Romanization of Chinese (referred to as pinyin) or know many, if any, words in western languages such as English. Thus, digits are used instead, since they are universal and thus familiar.

**3. Probabilistic Context-Free Grammars (PCFG): an Overview**

A number of effective offline password cracking models exist today. There is, for example, a dictionary attack, which simply takes a wordlist and a number of mangling rules, and then applies those rules on the wordlist, outputting all the variations it creates. Another possibility is to use Markov models, which is essentially building up probability tables of what character (or characters) is most likely to follow one, two, or more characters. Generation is accomplished by recursively building from the most likely possibilities, given this set of known characters. The model this paper will study is the probabilistic context-free grammars approach.

**3.1** **PCFG Background**

The PCFG approach is dependent on the idea that not all password guesses are equally likely to succeed. What this essentially means is that there are certain structures, and subsequently certain guesses, that are more likely to succeed as a password. Formally, a context-free grammar revolves around production rules between nonterminals to either terminals or another nonterminal. In relation to text, nonterminals can be parts of speech, while terminals can be the words themselves. For example, a highly likely context-free grammar would be noun then verb. The nonterminals are noun and verb here, and they can be substituted by words from a list of nouns and a list of verbs, respectively, to generate possibilities. For this paper, context-free grammars will be applied to passwords, and what that means is that character classifications will be looked at rather than parts of speech. Usually, when looking at a password, each character’s type is looked at and noted, such as the password “hacker123@!” will be seen as a string of 6 alphabet characters (a), then 3 digits (d), ending with 2 special characters (s). With a structure like “a6d3s2”, a PCFG model will recursively choose the most likely “a6” candidates, followed by the “d3” then “s2” candidates. Each component (or nonterminal) of the structure is looked at independently, and not affected by what preceded or follows it.

**3.2 Basic Process**

The general process of the PCFG model can be broken down into two steps (much like the dictionary attack and Markov model approach): the training step and the testing step.

*Training Step:* Given an input file to train off of, the algorithm looks at every single password and determines its structure. As it is iterating through every element in the training set, it is also maintaining a set of structures that exist in this training set and keeps track of the frequency of each. By the end, a probability table of the likelihood of each structure existing can be compiled. In addition, the algorithm can record all of the different possibilities for each component and create probability tables of these at the end. For example, the algorithm is examining element “xpecial42!” and creates the structure “a7d2s1” for it. Before moving onto the next element, it can add “xpecial” into the list of “a7” possibilities, updating its frequency if need be, and perform the same action for “42” and “!”. Through this implementation, probability tables of candidates for each nonterminal in the structures can also be created.

*Testing Step:* In order to generate passwords based off of what the algorithm learned from the training set, it looks at the probability of the structures and combines them, in a sense, with the probabilities of the candidates from the different components, outputting generated passwords in a decreasing order. This generated passwords list can then be compared against the a testing list, in order to determine the accuracy rate of the generation, and thus how effective the algorithm is.

**4. Chinese-language Passwords’ PCFG Training: Analysis**

Due to the nature of Chinese language passwords, a little less than half of the passwords from both datasets contain only digits. Thus, for the purposes of a more substantial analysis, this study chooses to focus specifically on the passwords that contain alphabet characters. This does not mean that digit strings are wholly ignored, but rather, the generation of digits in the testing step will rely purely on a probability table of digit strings. More will be said about this in the Probability Tables section. Also, for the training portion of this PCFG approach, both the 70yx and CSDN datasets were split up into training sets and testing sets with a ratio of 4 to 1 (80% of the dataset in the training set; 20% in the testing set).

**4.1 Building a Parser for Pinyin and English Words**

Even though the vast majority of the passwords in the Chinese-language datasets used ASCII characters, it does not mean that the passwords with alphabet characters in them necessarily contain English words. In fact, it intuitively makes more sense that the majority of those would contain pinyin words instead. However, in Bonneau and Xu’s paper, they tested each password in the 70yx and CSDN datasets for whether or not it contained a pinyin word, and ultimately determined that approximately 15.9% and 14.5% of the passwords, respectively, contained pinyin. Even though this preliminary rate of passwords that contain pinyin seems relatively low, it is still worth analyzing and ensuring that the rates are correct. Thus, through the progression of this study, a parser determining whether or not a password contained English, pinyin, random, or a hybrid of English and pinyin words was built.

In order to determine whether parts of passwords are English or pinyin, it is first necessary to obtain a dictionary of English and pinyin words, as well as a frequency list of both, as there will naturally be words present in both dictionaries. The pinyin dictionary and frequency list was obtained from Jun Da at Middle Tennessee State University [4], which contains the frequency of 9,933 Chinese characters and also gives each character’s set of pinyin, as some characters can have more than one pinyin attributed to it. The English dictionary and its frequencies were taken from the Google Ngram Viewer. Specifically, all of the files for the 1-grams were downloaded and parsed to only contain the word and its frequency in the most recent year (2008 for most of the words). Then, it was pared down to contain only words that occur over 400 times. This modification was done to limit the size of the dictionary, thus helping the algorithm’s speed. While this is not ideal, in reality, this should have a very minimal negative effect, if any, as it contains all of the *common* English words, including English proper nouns. With these dictionaries and frequency lists properly parsed, the algorithm takes them and loads them into separate tries (one for English, another for pinyin).

**4.2 Classification**

For this analysis stage, it is vital for the algorithm to determine what kinds of strings make up the password. For example, the algorithm should be able to detect that a potential password of “leadme32” is composed of two English words: “lead” and “me;” rather than 3 Chinese pinyin words: “le,” “a,” and “me.” In regards to classifying if a password is primarily English, pinyin, a hybrid, digits, or random, a basic structure for each password is first built. Then, as long as the password contains alphabet characters, the algorithm will recursively search both the English and pinyin dictionary tries and collect all possible English and pinyin words that are present in the password. After it has these two lists, provided that either of them is not empty, the algorithm then continues onto reorder the candidate words from both dictionaries by index within the password. Then it builds all the possible combinations of words between the English and pinyin lists, keeping track of a value, which this study will call “score,” attributed to each possibility. This score is obtained by taking the logarithm of the probability of each word in the combination, scaled according to its Shannon entropy value (depends on if the word if from the English or pinyin dictionary), and then summing them all together. In the end, the password is determined to be made up of the words in the combination with the lowest score.

**Example Walkthrough:** Suppose the password the algorithm is examining in this instance is: “woaibread19”. In the first part of the algorithm, it creates the list of potential English and pinyin words that could have made up this password. For the sake of this example, the lists will be shortened from what they actually will be in reality. English list: i, bread, read; pinyin list: wo, a, ai, re, e, a. The next step is ordering all of the potential words, so the order here would be: wo, a, ai, i, bread, read, re, e, a. Then, the algorithm iterates through every word in this ordered list, and recursively builds up a possible combination, using the rest of the words in the list. One pathway it can take is to look at “wo,” then choose “ai” rather than “a.” At this point, it must skip over “i", since that possibility has been used up, essentially, by the word just added. The next possibility is “bread,” which it uses, and now it is out of potential words, since “bread” uses up the rest of the letters in this password. In this case, the pathway is the correct one, and it accurately decides that the password is made up of 2 pinyin words (“wo” and “ai”) and one English word “bread.”

This determination of what kinds of words make up the character strings of a password is necessary in order to compose the second layer of this PCFG model. Rather than sticking purely with looking at grammatical structures of what kinds of characters the password is composed of, this paper wishes to examine if knowing what *kinds* of words within the structures will provide a more effective generation scheme when testing. Thus, this parser provides the necessary information in order to build these structures. After it has decided what combination of words is the best for a password, if there even is any, it composes the new structure with nonterminals consisting of digits, special characters, pinyin, English, and random. Random in this case indicates a string of alphabet characters that could not be classified as either pinyin or English.

**4.3 Probability Tables**

Once the best combination for the password has been decided by the parser, the algorithm also inputs those words, as well as the digits and special characters from the rest of the password if they exist, into their respective probability tables. Going back to the example given above, for password “woaibread19”, the words “wo” and “ai” would be added to a table of pinyin consisting of only 2 characters, while incrementing the frequency count of each if they already exist. The same thing is done for the word “bread” in the list of English words of length 5. Finally, the digit string “19” is added to the table of 2-digit strings. Nothing additional is done to strings of digits or special characters; it is a simple addition to the proper table.

**5. Chinese-language Passwords’ PCFG Testing: Generation**

The generation phase of this algorithm is very familiar to a typical PCFG generation, and a walkthrough of this process will be given below. Initially, the procedure loads in the probability tables that were created from the training phase. This will always consist of the grammatical structures, and almost always pinyin, English, random, digits, and special characters. The only times one would not exist is if there were no instances of them in the training set. Once the probability tables are loaded, the algorithm computes each of the grammatical structures’ initial probabilities and puts them onto a priority queue, where it prioritizes the maximum value. What this means is that for a certain structure, its initial probability value will be the value of the probabilities of each nonterminal’s most likely candidate, multiplied together, relative to the best probability for this structure. The starting probabilities for each structure will always be the best, so they will all be scaled to one, while the subsequent probabilities for the structures will be scaled accordingly. With these initial values on the priority queue, the algorithm enters into a loop until a certain generation threshold is reached or until the priority queue is emptied. In each iteration of the loop, the algorithm removes the highest priority grammatical structure, and grabs current most likely probabilities, and recursively builds up passwords from those possibilities. Once those possibilities has been depleted, the algorithm computes the next most likely probability value, and then reinserts the grammatical structure with this new probability back into the priority queue.

**Example Walkthrough:** Suppose the probability tables are:

*grammatical structures n6 e4 p3*

“n6”: .6 123456: .9 “love”: .8 “hen”: .5

“e4p3”: .4 000001: .1 “duck”: .2 “xia”: .5

For the “n6” structure, the best probability it can achieve is .9; for the “e4p3” structure, the best probability that it can achieve is: .8 \* .5, which is .16. Thus, the initial elements and their values put on the priority queue would be: (“n6”, .6 \* 1 = .6) and (“e4p3”, .4 \* 1 = .4). The first element taken off the queue would be “n6”, and it would generate “123456,” since there is only one candidate with probability .9 in the “n6” probability table. Then, the algorithm looks for the next best probability for “n6”, which happens to be .1 in this case. Since this .1 will be compared against the best relative probability, it will be scaled according to the .9, so it becomes 0.1111, thus when putting this back on the priority queue, “n6” will be paired with a value of 0.6 \* .1111 = 0.0666667. At the next iteration, the grammatical structure to be taken off the priority queue will be “e4p3,” since it has a value of 0.4. By structuring it this way, the algorithm ensures that even though certain structures may be more likely than the others, if the candidates for the nonterminals of the structure are highly unlikely, then the other grammatical structures are given a chance for generation, since the passwords generated from them may be more likely to be a successful hit against the test list. This process is completed until a threshold is reached or until the priority queue is empty.

**6. Results**

Preliminary analysis of the passwords in the training files for 70yx show that the percentage of passwords that contain pinyin is only around 20.08% (the training set contained a total of 7,258,372 passwords, 1,457,629 of which were classified as having pinyin words). However, when that over 7 million is pared down to only passwords that actually contain a notable number of alpha characters (at least 3 or 4, depending on the length of the overall password), then that percentage rises to 87.13%.

The current OMEN algorithm relies on the relative probabilities of given a two-letter prefix, to find the most likely next character. These 3-grams represent the structures of passwords fairly well, but perhaps can be bettered. At this point, this paper explores the idea of modifying from 3-grams to 5-grams to test if the latter will more accurately resemble the patterns inherent in human created passwords.

**4.1 5-grams Implementation**

In the normal implementation, the generation depends on 3-grams; but in this modified OMEN algorithm, the generation will try to depend on 5-grams as much as possible, reverting to 4- and 3-grams when necessary. Overall, there will be a total of eight probability tables generated: five absolute, three relative. The absolute tables will range from 1-gram to 5-grams, while the relative tables will range from 3-grams to 5-grams. All the tables from the original algorithm are generated, and the rest of the tables are generated in the way detailed below.

Look at the previous level of absolute or relative grams’ table (depending on what kind of table is being built), and for each element in that list, find all possibilities within the 3-grams relative table, given the prefix from the last two characters. Then, find the probability by multiplying the probability from the previous level’s table by the probability in the relative table. For example, when generating the absolute 3-grams table, look at each element in the absolute 2-grams table (suppose the first element is ‘de’), and then look up the prefix involving the last two characters in the 3-grams relative table and add each possibility to the new table (so ‘ded’, ‘deb’, ‘def’, etc. may be added).

With the addition of the tables, the only difference in the generation of passwords involves choosing from the highest possible grams table each time. For example, when generating passwords of length eight, the old algorithm would first start off with all possibilities from the 2-grams absolute table, and then repeatedly generate from the relative 3-grams table until the length requirement is met (in this case, the generating of letters from the 3-grams table happens six times). But with the new algorithm, it would first grab from the 5-grams absolute table, and then for the rest of the generation, take the last two characters from the 5-grams absolute table string and find it within the relative 5-grams table.

**4.2 Proposed Benefits**

By expanding to 5-grams overall, the algorithm operates faster because more of the work was transferred to the pre-computation stage of building the tables. In general, determining the possible level combinations that will sum up to a certain level for the generation phase can be lengthy, especially as the number of tables to go through increases. With the old algorithm, for generating all the possible length-level combinations for passwords of length 8, level 30, the number of combinations grows to over 300,000. But with the new algorithm, that number is cut down to 31. Once all the combinations are determined, then can the passwords be generated.

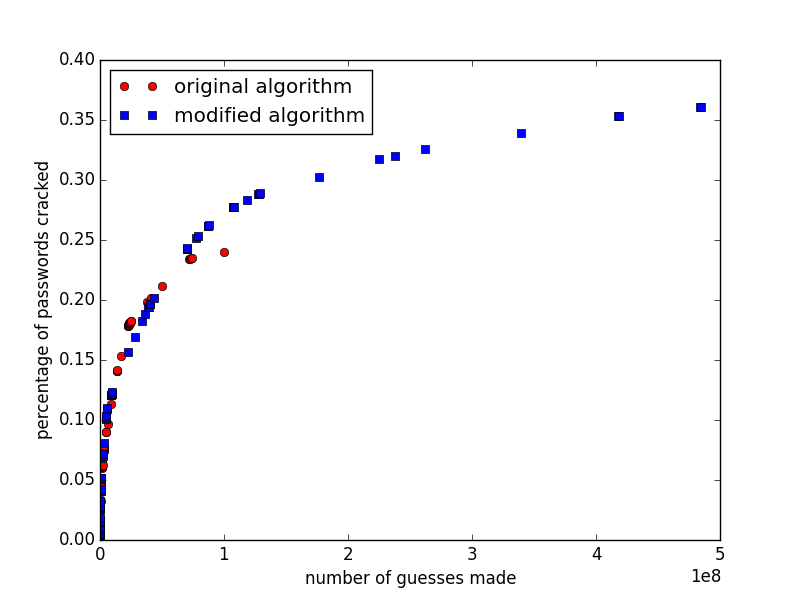
With this decrease in the number of tables to generate from, the total number of levels for each probability table can be increased. By increasing the total number of levels from 10 to 20 (or more), the overall granularity is finer and thus can benefit the generation phase. This is from reaching the more probable passwords earlier in the generation process because with the probabilities divided into simply ten levels, each level has to contain a wider range of probabilities whereas by increasing the range of levels, the generation of passwords where the levels are lower will more likely have a higher success rate.

**5. Results**

Below are plots of the percentage of passwords that are successfully cracked, by both the original algorithm and the modified algorithm, after one billion guesses.

**5.1 MySpace Dataset**

In the paper by Dürmuth, et al [1], the algorithm was run on a list of approximately 50,000 leaked MySpace passwords. Their experiment consisted of using 30,000 words in the training set, and 20,000 words in the testing set. In this paper, however, the list found of MySpace passwords [5] only numbered up to around 37,000, so 20,000 was used for the training set, and the remaining ~17,000 for the testing set.

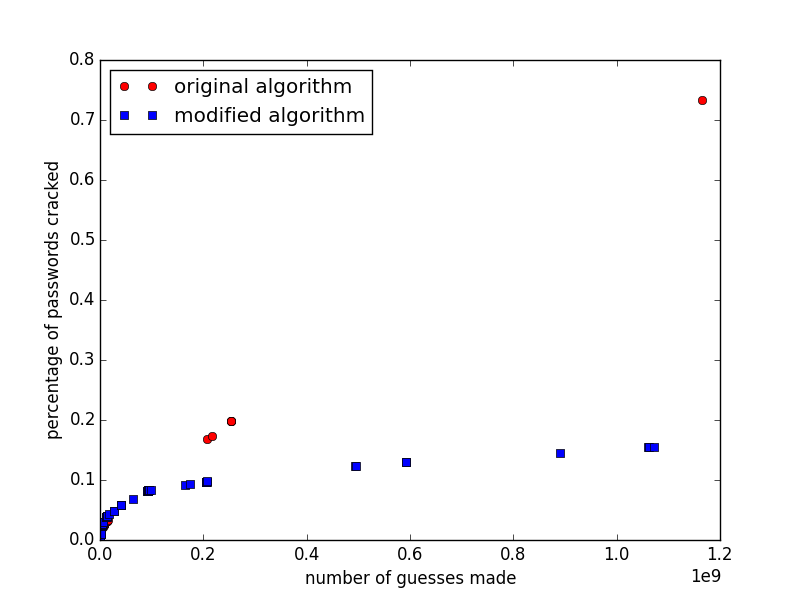
****

As can be seen on the plot above, approximately 500 million guesses were made rather than one billion because after that point, all of the estimated probabilities decreased down to zero, so there would be no further improvement in the percentage of cracked passwords. It could be because of the smaller number in the training set that the success rate is lower than the 68% purported by the paper.[[2]](#footnote-2) In fact, with this new method of choosing lengths and using the success probability from the sample as an indicator of the overall success, it is important to achieve as many length-level combinations with non-zero success probabilities as possible. With a larger dataset to train from, this test could potentially have generated up to one billion guesses while still showing improvement in the success rate.

From running the two algorithms on this MySpace dataset, it appears that the new algorithm outperforms the older one by a slight margin. The old algorithm flattens out around 25%, while the new one at 35%. In this case, the new algorithm’s method of basing as much of the generation off of 5-grams as possible proves to be beneficial.

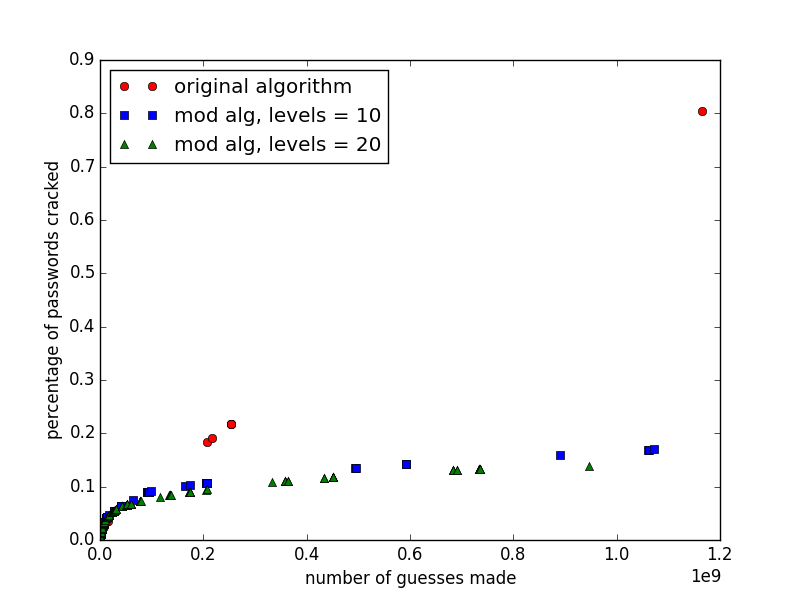
**5.2 Rockyou Dataset**

The Rockyou list of leaked password is notable for its large size – contains 32.6 million passwords – thus making the overall Markov model more robust. In the original experiment, the Rockyou list was split into a training set of 30 million, and then a testing set of 2.6 million. In this version of the experiment, however, the training set is scaled down to 3 million in order to ensure the tables generated remain a manageable size and that the time spent on generation remains reasonable.



The old algorithm clearly outperforms the new one in this experiment. Everything was kept the same between the original experiment and this current one (including limiting the number of levels for the probability tables to 10), except for the sizes of the training and testing sets. The low success rate (flattening at 20%) of the modified algorithm may be attributed to this discrepancy because the reliance on 5-grams within the Markov models means it is even more vital for the Markov models created from the training set be as accurate and as representative of the patterns within passwords as possible.

What is interesting to note is that the decreased number of elements in the training set does not appear to have a negative effect on the old algorithm. Due to the nature of this length and level enumeration algorithm, the success probability of a sample of approximately 100,000 elements for the length-level combination, is multiplied by the total number of passwords possible for that combination. For certain length-level combinations, the number of possible passwords there are can amount up to the hundreds of millions, and in one case, close to one billion (the jump in points between .3 billion guesses to 1.2 billion guesses is due to this); thus, even with a lower success probability, the number of cracked passwords would still result in being rather high (for example, the jump between 20% to 70%).



This experiment was repeated except the new algorithm was employed with the number of levels ranging from 0 to 19 (so 20 total), rather than 0 to 9, to test the granularity effect. However, rather than seeing an improvement in percentage of passwords cracked, there is a small dip.

**5.3 Sources of Error and Methods for Correction**

One of the most major sources of error comes from the differing training and testing datasets used within the experiments. In the paper, the MySpace dataset consisted of 50,000, while the MySpace dataset found here was around 37,000. In the second set of experiments, the training set consisted of significantly less than the original paper’s training set, for the sake of conserving space and time, which could have led to the creation of inadequate Markov models, particularly when they number up to 4-, then 5-grams. The best way to fix this is, of course, to ensure that the datasets are the same. In the case of the Rockyou list, space and time would have to be sacrificed in order to successfully build the table probabilities and then to generate through the possibilities.

Aside from the differences in datasets, the other most significant source of error comes from the success probabilities derived from samples of each length-level combination. These samples are generated randomly, in the sense that the combinations of levels within each length-level is iterated through randomly, until the threshold number of passwords is generated. For certain length-level combinations, the success probability may be much higher or lower compared to the *real* success probability. Should the success probability from a sampling be significantly different from its real value, and the total number of passwords that are possible from that combination is extraordinarily high (such as in the near one-billion case), then the results may be considerably skewed. For this error source, the most reliable way to fix this would be to simply generate *all* the possibilities at each length and level, compute the success probabilities, and then plot accordingly. This, of course, would also require significantly more time. A less time consuming option would be to generate several rounds of these probabilities, and then average it out to obtain a more accurate idea of what the success probability should be.

**6. Conclusion**

In this reproduction of the work done by Dürmuth, et al [1], their Markov model approach to generating passwords is confirmed to be highly effective in being able to crack passwords at rates of 60-70%. Another area that was explored in this work was modifying the generation scheme to be based off of 5-grams instead of 3, as well as to modify the total number of levels to sort the probabilities into. In the case of using 5-grams, for the Rockyou dataset, even though the 3-grams and 5-grams were developed from the same three million training set, it is clear that the 5-grams do not generate successful passwords as well. Furthermore, based off of these experiments, it is shown that increasing the granularity of the levels does not benefit the success rate either. However, these results are not conclusive, and it is very much worth exploring further, with more time and space. In fact, as a continuation of this study, running several trials of this experiment with the same training and testing sets would be beneficial in accurately determining the success probabilities of the length and level combinations. That, combined with expanding the number of items in the dataset to ensure a better trained Markov model, would lead to a reliable conclusion of the efficacy of 5-grams and increased granularity.

**7. References**

1. Honan, Mat. Kill the Password: Why a String of Characters Can’t Protect Us Anymore. *Wired,* 2012. http://www.wired.com/2012/11/ff-mat-honan-password-hacker/all/
2. Peterson, Andrea. China has almost twice as many Internet users as the U.S. has people. *The Washington Post*, 2014. http://www.washingtonpost.com/blogs/the-switch/wp/2014/01/31/china-has-almost-twice-as-many-internet-users-as-the-u-s-has-people/
3. Of contrasenas paper.
4. http://lingua.mtsu.edu/chinese-computing/statistics/char/list.php?Which=MO
5. Dürmuth, M., Chaabane, A., Perito, D., and Castelluccia, C. When Privacy meets Security: Leveraging personal information for password cracking. INRIA 2013.
6. Hawes, John. Only 24% of Europeans use different passwords for different websites. *Naked Security*, 2013. http://nakedsecurity.sophos.com/2013/11/27/only-24-of-europeans-use-different-passwords-for-different-websites/.
7. Marechal, Simon. Automatic mangling rules generation. *Openwall*, 2012. http://www.openwall.com/presentations/Passwords12-Mangling-Rules-Generation/.
8. Dell’Amico, M., Michiardi, P., and Roudier, Y. Password Strength: an Empirical Analysis. Eurecom. http://www.eurecom.fr/en/publication/2910/download/rs-publi-2910\_1.pdf.
9. MySpace Passwords. https://wiki.skullsecurity.org/Passwords.

1. Information on both sites is provided by Bonneau and Xu’s paper. [↑](#footnote-ref-1)
2. page 8 of [1]. [↑](#footnote-ref-2)