[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.

**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 “6a3d2s”, a PCFG model will recursively choose the most likely “6a” candidates, followed by the “3d” then “2s” candidates. Each component 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.

Using this technique, the emphasis is placed on the structure of passwords themselves, such as the positions of uppercase and lowercase letters, numbers, and special characters. Similar to John the Ripper’s dictionary attack process, the probabilistic context-free grammars approach needs to be fed a list of passwords in the training phase, to work off of. It looks at every single password and makes a note of the structure, keeping track of which ones occur the most frequently. Then, in the attack phase, it goes through the structures in order of decreasing probability. For each structure, it looks at the continuous runs of letters and plugs in words fitting the length from a dictionary; and then for the numerical digits and special characters, it is substituted by certain numbers and characters according to what was present during the training phase. The reasoning behind this algorithm is to guess passwords consisting of six letters then two numbers faster than passwords consisting of two letters, three numbers, and three letters, because some structures occur more frequently than others. Overall, the success rate of using this method can reach up to 48%, but is frequently lower [4].

**2.3 Markov Models**

Markov models are, naturally, an effective way to crack passwords. The reasoning behind this is directly tied to the idea that as humans, the passwords we create are not entirely random. In fact, many passwords follow a specific pattern and/or contain actual words. Because the characters in passwords are typically not independently chosen, thus not completely random, they usually depend upon the character (or characters) before them. For example, the three-letter string ‘can’ is much more likely to exist in passwords than ‘cay’ or ‘cqn.’

In order to implement this method involving Markov models, one needs to calculate two kinds of probabilities: a set of initial, absolute, probabilities, and a set of transition, relative, probabilities. The differences between these two are that the absolute probabilities are calculated purely from the list of passwords used for training. For example, in calculating the absolute probabilities of 3-grams (strings of length 3), one finds all the different 3-grams in the list and keeps track of the frequency of each, thus the probability. On the other hand, the set of relative probabilities keeps track of all the different 3-grams, much like absolute, except instead of calculating the probabilities based purely off of the frequency of the 3-grams, it calculates based off of the previous two characters. What this means is that for a 3-gram like ‘cod,’ one divides the frequency of ‘cod’ by the frequency of all the ‘co-’ strings. These relative probabilities make up the core of this Markov model approach.

The other essential part of the Markov model method is to enumerate through the generated passwords in the proper order. Proper means to enumerate through them in a way that will optimize the success rate by the number of guesses made. Finding the proper enumeration method can be difficult because while one can calculate the probabilities, it is not as simple as putting all of the probabilities into a list and then enumerating through them in decreasing order. Thus, it is vital to figure out a proper enumeration technique, which is where OMEN comes into play.

**3. Ordered Markov ENumerator (OMEN)**

It is vital for an algorithm to enumerate through the generated passwords in an order that will hit upon successful passwords earlier in the generation phase than later. Should the process be limited to a number, such as the first million guesses, the passwords of higher probability need to be reached in the beginning, or at least within, that one million, rather than at the end, otherwise the number of passwords cracked would be needlessly low. Enumerating in a logical, probabilistic order speeds up the entire guessing process, and can give one a reliable idea of just how effective the algorithm is without having to go through literally every single one of the possibilities generated.

**3.1 The Process**

The most crippling flaw of the sorting purely by probability of the generated passwords is that one would need to generate *all* of the passwords in order to obtain the success probability of that length. It is only with those two pieces of information is one able to effectively enumerate the passwords’ probabilities in decreasing order. However, generating all of the passwords is a waste of time because some combinations created by the generator will be highly unlikely while some are extremely likely. Thus, it is important to find a way to enumerate through the generation in an effective fashion.

In Dürmuth, et al’s paper [1], they explain a method of enumeration where all of the probabilities are separated into a set number of levels, and then in the generation process, those levels are iterated in order from most likely to least likely, depending on an overall level sum. Breaking this down into pieces, the initial probability tables’ generation involves creating three tables: two absolute, one relative. The two absolute tables for one- and two-grams are created by looking at the password list provided and keeping track of the frequencies of each unique one-gram (and two-grams), and then dividing each by the total number of grams to determine the probability. The relative probability table is generated by first finding all of the unique three-grams and then for each group of three-grams with the same first two characters (the same prefix), determine the probability by taking the value of that unique element’s frequency over the sum of the frequencies of all the elements in that prefix group.

Then, for all of the values found in the absolute and relative probability tables, those numbers are discretized by taking the logarithm of the probability, and then applying a transformation in a way that each probability can be sorted into a number between 0 and -9. In the paper, they limit levels to ten, and use the syntax of the more negative the level number, the less likely it is. However, in this replication of their procedure, I eliminate the negatives, so that 0 still denotes the most likely level, and 9 the least likely. The formula to determine the levels is: . *c1* and *c2* are chosen such that the *n*-grams of the highest probability are given level 0, and the ones that are least likely given the maximum level number.

The enumeration and generation phase starts with picking an overall level and length, and then finding all possible combinations for that level. Afterward, it first looks at the absolute table of length 2, and grabs all strings with a level matching the first number in the combination array. Then, it looks at the relative table, and finds all the possibilities matching both the level at the next index in the combination array as well as fitting the prefix of the previous two characters. This continues on until the overall length generated reaches the given length.

**Walkthrough:** passwords of length 3 (l = 3) for a subset of the alphabet.

String Level String Level String Level

th 0 th -> e 0 tv -> i 1

tv 1 th -> i 1 tk -> b 0

tk 2 th -> r 2 te -> r 0

te 0 tv -> w 0 te -> b 1

For finding passwords of level 0: the only combination with two indices making an overall sum of 0 would be (0, 0), so the strings generated would be: ‘the’ and ‘ter’.

For finding passwords of level 1: the combinations making 1 would be (1, 0) which means the strings generated would be ‘thi’, ‘tvw’, and ‘teb’.

For finding passwords of level 2: the combinations making 2 would be (2, 0) and (1, 1), so the strings generated would be ‘thr’, ‘tvi’, and ‘tkb’.

This process would continue on for levels 3 and 4.

The other facet to this algorithm is choosing the proper lengths to enumerate through. For example, this means that one should generate more passwords of length six or eight than passwords of length four or seventeen, because the former lengths are more likely, thus more worthwhile to generate. However, going purely off of how frequently certain lengths appear in the training set could lead to incorrect results. This is due to the idea that certain passwords in unlikely lengths are still more likely to be successful than passwords at more likely lengths. Rather than iterating through lengths completely, based on frequency, it is more advantageous to jump between lengths throughout the generation process.

The method for choosing lengths put forth by Dürmuth, et al. [1] is to, for all possible length values, generate all passwords for that length at level 0, and then compute the success probability *sp*l,0. The success probability is determined by the number of passwords that actually match passwords in the testing set over the total number of passwords generated. Afterward, all the elements are put into a list in the format of (success probability, level, length), which is then sorted in descending order by probability. Then, this process is repeated:

1. Remove the element at index 0, which represents the length and level with the highest success probability.
2. Generate all the possible passwords at the given length from the removed element, but at a level incremented by one. After this generation, the success probability for that length and level combination is thus computed and then inserted back into the list. For example, suppose the element at index 0 has a length of 6 and a level of 0. That means the generator would enumerate through passwords at length 6, level 1. The success probability is computed for this combination, and then put back into the overall list.
3. Sort the list of elements including the success probability, and then repeat from step one until the list is exhausted or enough guesses have been made.

**3.2 Implementation Changes**

In this replication of the Dürmuth, et al. [1] algorithm, the generation of passwords for specific length-level combinations is identical to the one put forth in the paper, except rather than using negative levels, this version uses positive. In the determination of what order the length and level combinations should be iterated through, however, this method differs. In these experiments, rather than starting off the list with simply the lengths at level zero, this algorithm goes through all of the possible lengths and generates a minimum of *n* passwords, then computes the success probability of each. Once all those probabilities are calculated, it goes through and finds the total number of guesses that *could* have been made at that length and level, rather than cutting it off after reaching *n*. The reasoning behind this is that the algorithm described in the paper is essentially using the previous level’s success probability as an indicator of the next level’s success probability. Essentially, the probabilities that are put onto the list to determine what the next length-level combination to enumerate through should be, may not be representative of the actual success probability of the next level for the same length. With this current method of choosing lengths, it looks at the success probability from a suitable sample of each length-level combination, and then chooses the lengths and levels accordingly.

**4. Modified OMEN**

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.

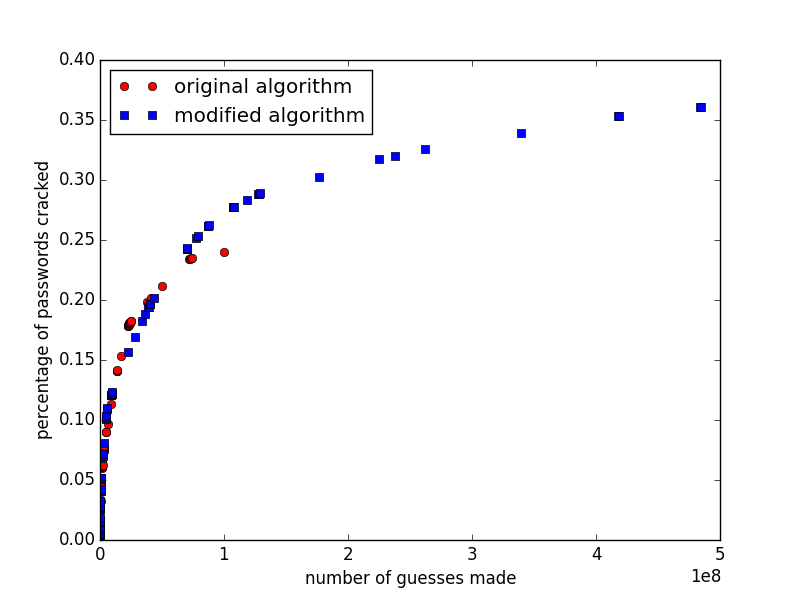
With this decrease in the number of tables to generate from, the total number of levels for each probability table can be increased. By increasing the total number of levels from 10 to 20 (or more), the overall granularity is finer and thus can benefit the generation phase. This is from reaching the 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.

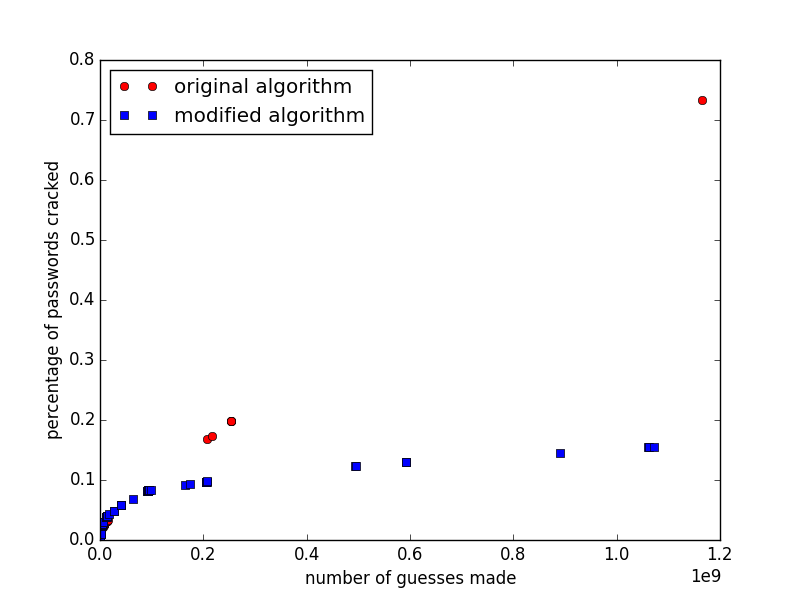
****

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.

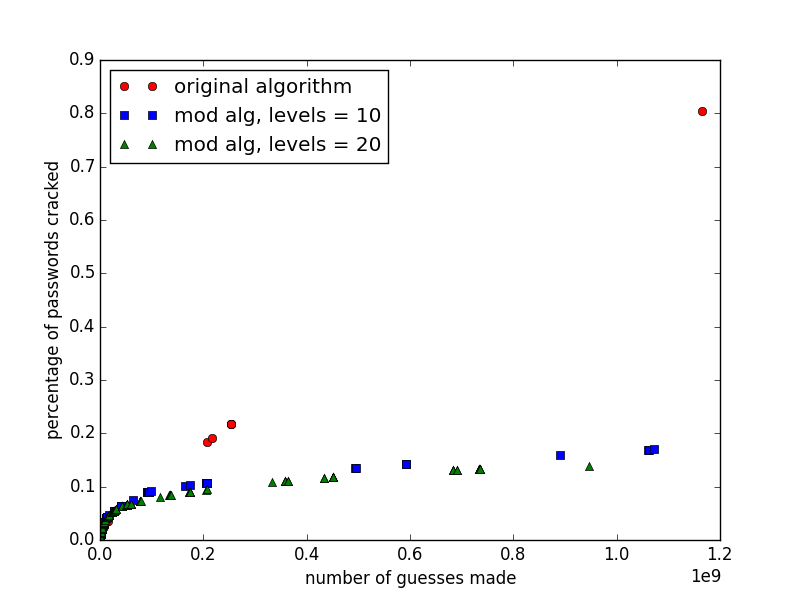
**5.2 Rockyou Dataset**

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. Dürmuth, M., Chaabane, A., Perito, D., and Castelluccia, C. When Privacy meets Security: Leveraging personal information for password cracking. INRIA 2013.
5. 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/.
6. Marechal, Simon. Automatic mangling rules generation. *Openwall*, 2012. http://www.openwall.com/presentations/Passwords12-Mangling-Rules-Generation/.
7. 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.
8. 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)