An Examination of 密码: Utilizing the Probabilistic Context-Free Grammars Approach in Guessing Chinese-language Passwords

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, analyze the prevalence of pinyin and English words within them, and determine if using the probabilistic context free grammars (PCFG) model leads to a more effective way to crack the passwords, compared to a typical password cracker (John the Ripper). Ultimately, the findings cannot conclusively state that the PCFG approach has more success in finding passwords, though it does hint that the PCFG model may be effective, provided a few additional modifications.

An Examination of 密码: Utilizing the Probabilistic Context-Free Grammars Approach in Guessing Chinese-language Passwords

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. Already, there have been numerous studies done on the strength of a password, and subsequently, just as many papers published on the topic. Some of the most common techniques used, when it comes to offline cracking, are: brute force, dictionary attacks, Markov models, and probabilistic context-free grammars (PCFG).

1.1 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 Bonneau and Xu's paper on the problems with password encoding on the web [3]. Within it, they examined two leaked Chinese-language datasets: 70yx and csdn. A brief overview of the sites is given below[1].

[1] Information on both sites is provided by Bonneau and Xu's paper.

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[2]. 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-

[2] These statistics all come from Bonneau and Xu's paper, [3].

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[3]. 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

[3] Again, statistics are from Bonneau and Xu, [3].

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 (a visual of this is given below, in figure 1). Each component (or nonterminal) of the structure is looked at independently, and not affected by what preceded or follows it.

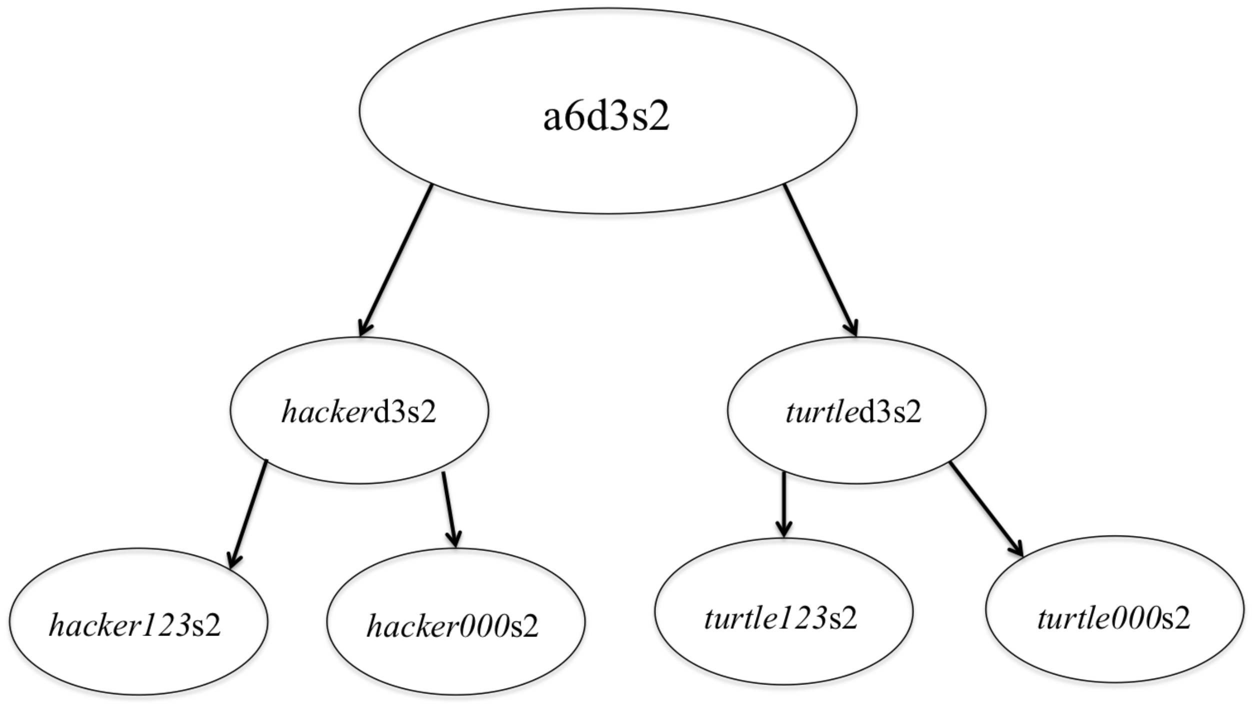

Figure 1: the parts in non-italics are the structures. Starting from `a6d3s2`, it can fill in `a6` with `hacker` or `turtle`. Both are pathways that will lead to a string generated with the root structure.

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 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 [3]. 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 [5]. 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.

$$\text{score for password } p\text{: } \frac{\sum_{i=0}^{k}(-1 \times \log_2 prob[w_i])}{S_i}$$

In the equation above, *i* is the index for the word in this specific combination; *w* denotes the word, and *S* denotes the appropriate Shannon entropy value, according to the type of the word.

Ultimately, 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 (`d`), special characters (`s`), pinyin (`p`), English (`e`), and random (`r`). 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 empty. In each iteration of the loop, the algorithm removes the highest priority grammatical structure, and grabs the current most likely probabilities, and recursively builds up passwords from those possibilities. Once those possibilities have 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%. Thus, even though the passwords with pinyin do not necessarily make up a large portion of the total number of passwords in the training set, they should still be considered as a noteworthy subset of the whole, especially when relatively compared to the number of passwords that actually contain over 3-4 alphabet characters.

To determine the success of this PCFG algorithm, the datasets were benchmarked against John the Ripper's wordlist attack, using its default rules. This is essentially a dictionary attack, where the rules provided by John the Ripper (JtR) create variations of the words in the given wordlist. No additional modifications were made to JtR's generation process, as the goal here is to observe the difference in success rates between this PCFG algorithm and a typical password cracker that has had decent results on Western-language passwords.

6.1 70yx Dataset

The leaked 70yx dataset contains 9,072,966 passwords, which was divided into a training set of 7,258,372 passwords (approximately 80% of the total) and a testing set of 1,814,594 passwords (approximately 20% of the total). Unlike the 70yx dataset, the proportion of passwords which contain pinyin is much lower, consisting of 0.1600 of the total number of passwords in the training file. When compared primarily against number of passwords with at least 3-4 characters, the percentage increases to 0.5029.

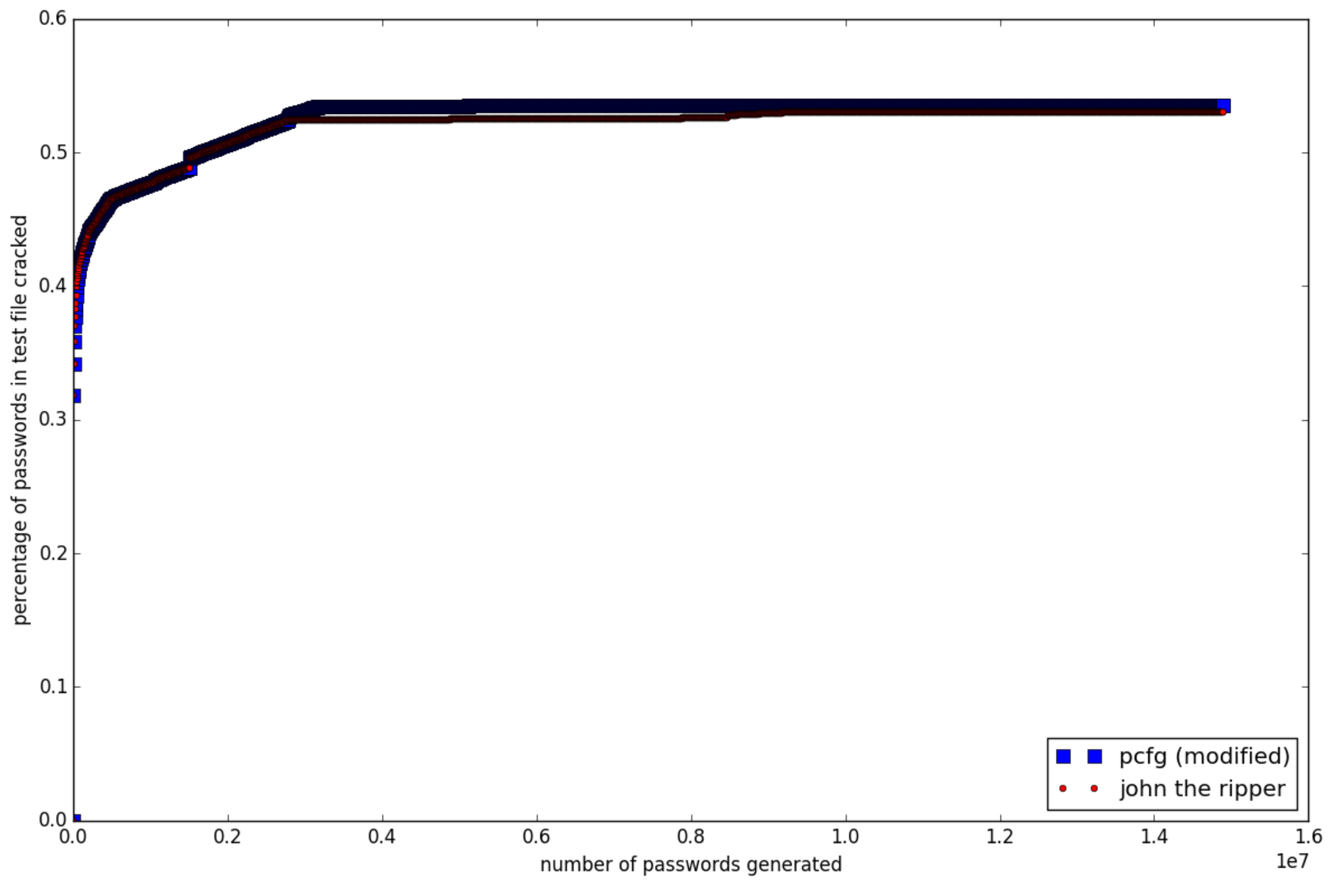

Figure 2: plot of the percentage of passwords successfully found from the 70yx test file against number of passwords generated total

As can be seen by the plot above, approximately 16 million passwords were generated by both the PCFG algorithm and JtR's wordlist attack. Around the 4 million mark, there is little increase seen in the accuracy rate for both generation methods. In fact, the plot shows that both algorithms follow each other very closely throughout the entire generation process, with very minor differences. Ultimately, the accuracy rate for the new algorithm s 0.5349, while the accuracy rate from JtR is 0.5302. There is a very small increase in the success rate of the PCFG approach, but this increase is not tremendously significant. In fact, even when approximately 100 million passwords were generated by using the new algorithm, the success rate still could not break 54%, hovering around 0.5358.

6.2 CSDN Dataset

The CSDN dataset is somewhat smaller than 70yx, containing only 6,428,630 passwords. These passwords were divided into a training set of 5,142,904 and a testing set of 1,285,726 passwords.

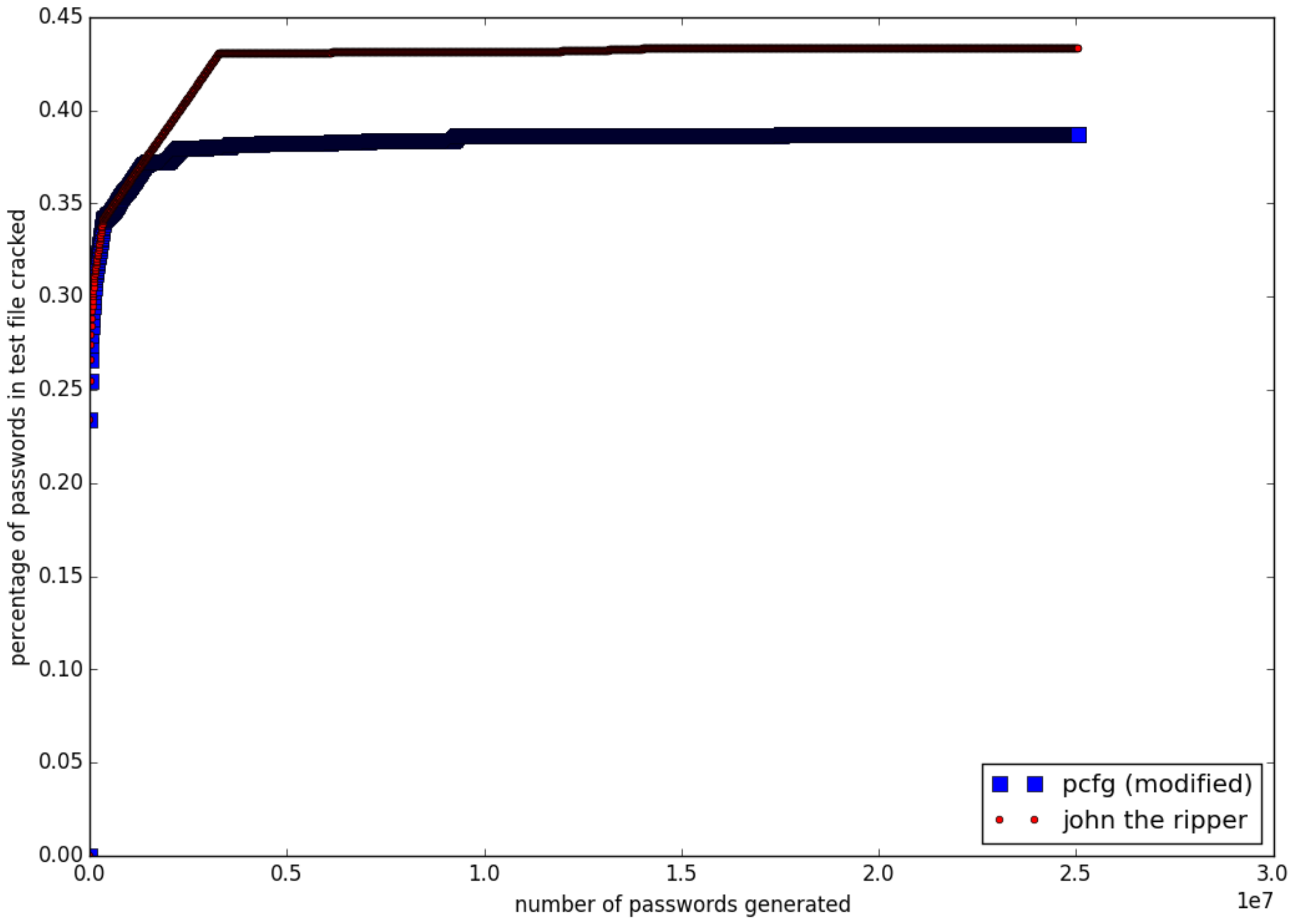

Figure 3: plot of the percentage of passwords successfully found from the CSDN test file against number of passwords generated total

Unlike with the 70yx dataset, which saw a very minor increase in accuracy rate between the PCFG algorithm and JtR's wordlist attack, for the CSDN dataset, there is a very clear difference in accuracy between the two, with JtR outperforming the new algorithm. Both generate around

25 million passwords, but whereas JtR’s accuracy rate reaches .4334, PCFG’s peaks at 0.3871. Even when the PCFG algorithm is allowed to run for a longer amount of time, generating around 100 million passwords, it still does no better than 0.3885.

6.3 Sources of Error and Methods for Correction

The step with the highest probability for error is in the construction of the grammar structures. This is due to the fact that the grammar structures rely on the parser deciding if a password contains English or pinyin words, a hybrid of both, or has mainly random alpha characters. If the parser incorrectly classifies a high number of passwords incorrectly, then the probability table of the grammar structures will naturally be incorrect as well. The parser was sanity checked against a few hundred random passwords, but even then, short of having a person double check each classification that is made, there is always the potential that the parser classifies incorrectly. Unfortunately, there is no clear way around this other than a more robust system for sanity checking.

Another potential source of error would be what passwords were included into the training set of passwords, as that was chosen randomly. Due to the high number of passwords in the training set, there should not be any glaring anomalies or absences of a certain kind of password from the set, but it is technically still possible. Also, the reliance on the training set as the sole source of words and strings is problematic to the algorithm overall. This will be discussed below.

6.4 Improving the Algorithm

At least with the 70yx dataset, due to the new PCFG algorithm not doing any worse than the JtR algorithm, and doing better, albeit very slightly better, that seems to give the indication that improvements can be made to bring the success rate of this algorithm significantly higher. Because of the worse performance with the CSDN dataset, which had fewer passwords that contained pinyin, the areas of focus should be in refining the generation of digit strings, random strings, and English words. Thus, the clearest way to improve the algorithm at the moment, which with more time would have been done, is to fill in the gaps in the probability tables. What this means is that in the training set, it is unlikely that every single possible permutation of digit strings would have been seen, nor every possible English word, and potentially not even every pinyin word (potentially in this case due to there being only 416 pinyin, compared to the over 130,000 English words in dictionary).

By relying purely on the training set as the source of words, random character strings, and digit strings in the generation phase, the algorithm is essentially neglecting anything *not* seen in the training phase. Most likely, many passwords in the testing set will contain words and character strings that were not seen at all in the training phase, yet those strings not yet seen should still be assigned a probability in order to give them a chance of surfacing in the generation phase. A significant modification to the algorithm can thus be made to hopefully improve the success rate: find all of the digit strings, English words, and pinyin words not present in the training set, and assign each of them a fixed probability. This probability can be derived from looking at the items on the probability table thus far and determining a cutoff point for when probabilities are low enough to signify that the presence of this string is not particularly significant. With all the leftover probabilities accumulated, they can all be divided evenly into the strings not present in the training set, to give them a chance of occurring in the generation phase.

The improvement capabilities can be seen through a simple example: suppose the structure `e4d3` is highly likely. Yet the only three digit strings encountered in the training phase were `123`, `000`, `423`, and `829`, with the latter having the smallest probability. This change in the algorithm would then ensure that all the other three digit strings, such as `242`, `890`, and the list goes on, all have a chance of occurring. If the structure is incredibly common, then the generation phase will get to a point where those strings will be used, and thus more successful guesses will likely be made.

Another potential source of improvement lies in adding another level to the nonterminals. At the moment, the structures only have one level of nonterminals; whatever components make up the structure will be replaced, without breaking them down further, by strings from another probability table. Rather than implementing the PCFG like this, it may be helpful to split up the initial nonterminals into smaller parts. What this means is that suppose the structure is `n6p3`, then after looking at all of the possible `n6` elements, the algorithm will then break `n6` down into `n3n3`, `n4n2`, `n2n4`, and so on, and then generating strings based off of that structure. By adding this other layer of nonterminals, rather than the algorithm moving onto another, potentially significantly less likely, structure, it can examine more likely strings which, when concatenated together, result in the original component. Going back to the example, after all the `n6` options are depleted, rather than going to the next structure, the algorithm will continue onto generate more technically `n6` strings, which are determined by compositions of smaller components.

Finally, in Bonneau and Xu's paper, they noted that the percentage of passwords containing keyboard patterns (such as `qwerty`) was much higher in Chinese-language passwords than Western-language passwords (11% compared to 3-5%, respectively) [3]. Knowing this, it may be worthwhile to modify the PCFG model to also include a nonterminal to handle strings containing keyboard patterns (usually involving adjacent keys). This would most likely improve upon the classification of random strings, since many strings may be considered random by the current algorithm, when really, they signify a keyboard pattern.

7. Conclusion

In this study of Chinese-language passwords and analysis of the success of applying an existing password-cracking model on it, the PCFG model appears to have potential for being very effective in guessing passwords correctly. At least with datasets that contained a significant number of passwords with pinyin in them, the new algorithm appears to outperform the existing John the Ripper's wordlist attack. It is for the same reason that can possibly explain why the algorithm had a lower accuracy rate for the CSDN dataset, which contained more passwords containing random strings and English words. Due to the smaller dictionary size of pinyin, without a more robust source for English words and random strings, the PCFG algorithm will struggle. Thus, the areas this study explored are very much worth investigating in more detail, provided more time and resources. As a continuation of this study, modifying the algorithm as described above would most likely lead to a significantly improved accuracy rate for the PCFG algorithm, and subsequently, it would be worth comparing against the JtR wordlist attack, loaded with a more robust set of rules (such as Korelogic's rules [6]). These modifications, combined with generating more than just 15-25 million passwords, would lead to a reliable conclusion of the efficacy of this PCFG algorithm on Chinese-language passwords.

7. References

1. M. Honan. 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. A. Peterson. 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. J. Bonneau and R. Xu. Of contraseñas, סיסמאות, and 密码: Character encoding issues for web passwords. In *Web 2.0 Security & Privacy (W2SP’12),* May 2012.
4. J. Da. Modern Chinese Character Frequency List. *Middle Tennessee State University*, 2005. http://lingua.mtsu.edu/chinese-computing/statistics/char/list.php
5. Ngram Viewer. *Google*, 2012. http://storage.googleapis.com/books/ngrams/books/datasetsv2.html
6. KoreLogic John the Ripper Rules. *KoreLogic Security*, 2010. http://contest-2010.korelogic.com/rules.html