2014 IEEE Symposium on Security and Privacy

A Study of Probabilistic Password Models

Jerry Ma
Quora, Inc.
ma127jerry@gmail.com

Weining Yang
Purdue University
yang469@cs.purdue.edu

Min Luo
Wuhan University
mluo@whu.edu.cn

Ninghui Li
Purdue University
ninghui@cs.purdue.edu

***Abstract*—A probabilistic password model assigns a probability value to each string. Such models are useful for research into understanding what makes users choose more (or less) secure passwords, and for constructing password strength meters and password cracking utilities. Guess number graphs generated from password models are a widely used method in password research. In this paper, we show that probability-threshold graphs have important advantages over guess-number graphs. They are much faster to compute, and at the same time provide information beyond what is feasible in guess-number graphs. We also observe that research in password modeling can benefit from the extensive literature in statistical language modeling. We conduct a systematic evaluation of a large number of probabilistic password models, including Markov models using different normalization and smoothing methods, and found that, among other things, Markov models, when done correctly, perform significantly better than the Probabilistic Context-Free Grammar model proposed in Weir et al. [25], which has been used as the state-of-the-art password model in recent research.**

I. Introduction

Passwords are perhaps the most widely used method for user authentication. Passwords are both easy to understand and use, and easy to implement. With these advantages, password-based authentication is likely to stay as an important part of security for the foreseeable future [14]. One major weakness of password-based authentication is that many users tend to choose weak passwords that are easy to guess. Addressing this challenge has been an active and important research area. A fundamental tool for password security research is that of *probabilistic password models* (*password models* for short).

A password model assigns a probability value to each string. The goal of such a model is to approximate as accurately as possible an unknown password distribution $\mathcal{D}$. We divide password models into two classes, *whole-string models* and *template-based models*. A template-based model divides a password into several segments, often by grouping consecutive characters of the same category (e.g., lower-case letters, digits, etc.) into one segment, and then generates the probability for each segment independently, e.g., [21], [25]. A whole-string model, on the other hand, does not divide a password into segments, e.g., the Markov chain model in [8].

Jerry Ma's work on this paper was done while working as a research assistant at Purdue University. Min Luo is with the Computer School of Wuhan University in Hubei, China; his work on this paper was done while being a visiting scholar at Purdue University. Weining Yang and Ninghui Li are with the Department of Computer Science and the Center for Education and Research in Information Assurance and Security (CERIAS) in Purdue University, West Lafayette, Indiana, USA. The first three authors are co-first authors.

We classify research involving password models into two types. *Type-1* research aims at understanding what makes users choose more (or less) secure passwords. To do this, one obtains password datasets chosen by users under difference circumstances, and then uses a password model to compare the relative strengths of these sets of passwords. *Type-2* research aims at finding the best password models. Such a model can then be used to evaluate the level of security of a chosen password, as in password strength meters. Such a model can also be used to develop more effective password cracking utilities, as a password model naturally determines the order in which one should make guesses.

Type-1 research has been quite active in recent years. Researchers have studied the quality of users' password choices under different scenarios [17], [18], [22], [23], [26]. In this area, earlier work uses either standard password cracking tools such as John the Ripper (JTR) [3], or ad hoc approaches for estimating the information entropy of a set of passwords. One such approach is NIST's recommended scheme [7] for estimating the entropy of one password, which is mainly based on their length.

Weir et al. [25] developed a template-based password model that uses Probabilistic Context-free Grammar. We use PCFG_W to denote this password model. In [24], they argued that entropy estimation methods such as that recommended by NIST are inaccurate metrics for password strength, and proposed to use the guess numbers of passwords. The guess number of a password according to a password model is defined to be the rank of the password in the order of decreasing probability. To compare two sets of passwords, one plots the number of guesses vs. the percentage of passwords cracked by the corresponding number of guesses in the testing dataset. Such *guess-number graphs* are currently the standard tool in password research. Plotting such graphs, however, requires the computation of guess numbers of passwords, which is computationally expensive. For many password models, this requires generating all passwords with probability above a certain threshold and sorting them. Some template-based models, such as the PCFG_W model, have the property that all strings fitting a template are assigned the same probability. One is thus able to compute guess numbers by maintaining information about templates instead of individual passwords. This is done in the guess calculator framework in [17], [20], which is based on the PCFG_W model. This framework uses parallel computation to speed up the process, and is able to go up to $\approx 4E14$ [20]. Even with this approach, however, one is often

DOI 10.1109/SP.2014.50

limited by the search space so that only a portion (typically between 30% and 60%) of the passwords are covered [17], [20]. Such studies thus miss information regarding the stronger passwords.

We observe that for type-1 research, in which one compares the relative strength of two sets of passwords, it is actually unnecessary to compute the guess numbers of all passwords. It suffices to compute the probabilities of all passwords in the datasets, which can be done much faster, since the number of passwords one wants to evaluate (often much less than 1E6) is in general extremely small compared to the maximum guess number one is interested in. We propose to plot the probability threshold vs. the percentage of passwords cracked curves, which we call *probability-threshold* graphs. In addition to being much faster to compute, another advantage is that such a curve shows the quality of passwords in the set all the way to the point when all passwords are cracked (assuming that the password model assigns a non-zero probability to each allowed password).

A natural approach for password models is to use whole-string Markov chains. Markov chains are used in the template-based model in [21], for assigning probabilities to segments that consists of letters. Castelluccia et al. [8] proposed to use whole-string Markov models for evaluating password strengths, without comparing these models with other models. At the time of this paper's writing, PCFG$_W$ is still considered to be the model of choice in type-1 password research.

We find this current state of art unsatisfying. First, only very simple Markov models have been considered. Markov models are known as n-gram models in the statistical language literature, and there exist a large number of techniques developed to improve the performance of such models. In particular, many smoothing techniques were developed to help solve the sparsity and overfitting problem in higher-order Markov models. Such techniques include Laplace smoothing, Good-Turing smoothing [13], and backoff [16]. Password modeling research has not taken advantage of such knowledge and techniques. Second, passwords differ from statistical language modeling in that passwords have a very definite length distributions: most passwords are between lengths 6 and 12, and there are very few short and long passwords. The n-gram models used in statistical language modeling generally have the property that the probabilities assigned to all strings of a fixed length add up to 1, implicitly assuming that the length distribution is uniform. This is fine for natural language applications, because often times one only needs to compare probabilities of sentences of the same (or very similar) length(s), e.g., when determining which sentence is the most likely one when given a sequence of sounds in speech recognition. For password models, however, assuming uniform length distribution is suboptimal. Finally, different password modeling approaches have not been systematically evaluated or compared against each other. In particular, a rigorous comparison of the PCFG$_W$ model with whole-string Markov models has not been performed.

In this paper, we conduct an extensive empirical study of different password models using 6 real-world plaintext password datasets totaling about 60 million passwords, including 3 from American websites and 3 from Chinese websites. We consider three different normalization approaches: *direct*, which assumes that strings of all lengths are equally likely, *distribution-based*, which uses the length distribution in the training dataset, and *end-symbol based*, which appends an "end" symbol to each password for training and testing. We consider Markov chain models of different orders, and with different smoothing techniques. We compare whole-string models with different instantiation of template-based models.

Some of the important findings are as follows. First and foremost, whole-string Markov models, when done correctly, significantly outperform the PCFG$_W$ model [25] when one goes beyond the first million or so guesses. *PCFG$_W$ uses as input both a training dataset, from which it learns the distribution of different templates, and a dictionary from which it chooses words to instantiate segments consisting of letters. In this paper, we considered* 3 *instantiations of PCFG$_W$: the first uses the dictionary used in [25]; the second uses the OpenWall dictionary; and the third generates the dictionary from the training set. We have found that the third approach works significantly better than the first and the second; in addition, all three instantiations significantly underperform the best whole-string Markov models.* Furthermore, higher orders of Markov chains show different degrees of overfitting effect. That is, they perform well for cracking the high-probability passwords, but less so later on. The backoff approach, which essentially uses a variable-order Markov chain, when combined with end-symbol based normalization, suffers little from the overfitting effect and performs the best.

In summary, the contributions of this paper are as follows:

- We introduce the methodology of using probability-threshold graphs for password research, which has clear advantages over the current standard approach of using guess-number graphs for type-1 password research. They are much faster to construct and also gives information about the passwords that are difficult to crack. In our experiments, it took about 24 hours to generate 10^{10} passwords for plotting guess-number graphs; which cover between 30% and 70% in the dataset. On the other hand, it took less than 15 minutes to compute probabilities for all passwords in a size 10^7 testing dataset, giving strength information of all passwords in the dataset. *We note, however, that for type-2 research, in which one compares different password models, one needs to be careful to interpret results from probability-threshold graphs and should use guess-number graphs to corroborate these results.*
- We introduce knowledge and techniques from the rich literature in n-gram models for statistical language modeling into password modeling, as well as identifying new issues that arise from modeling passwords. We also identify a broad design space for password models.
- We conduct a systematic evaluation of many password models, and made a number of findings. In particular, we

show that the PCFG$_W$ model, which has been assumed to be the state of the art and is widely used in research, does not perform as well as whole-string Markov models.

The rest of this paper is organized as follows. We introduce probability-threshold graphs in Section II, and explore the design space of password models in III. Evaluation methodology and experimental results are presented in Sections IV and V, respectively. We then discuss related work in Section VI and conclude in Section VII.

II. Password Models and Metrics

We use Σ to denote the alphabet of characters that can be used in passwords. We further assume that all passwords are required to be of length between L and U for some values of L and U; thus the set of allowed passwords is

$$\Gamma = \bigcup_{\ell=L}^{U} \Sigma^{\ell}.$$

In the experiments in this paper, we set Σ to include the 95 printable ASCII characters, and $L = 4$ and $U = 40$. Sometimes the alphabet Σ is divided into several subsets, which we call the character categories. For example, one common approach is to divide the 95 characters into four categories: lower-case letters, upper-case letters, digits, and special symbols.

Definition 1: A *password probability model* (password model for short) is given by a function p that assigns a probability to each allowed password. That is, a password model $p : \Gamma \to [0, 1]$ satisfies

$$\forall_{s\in\Gamma}\, p(s) \geq 0 \bigwedge \sum_{s\in\Gamma} p(s) = 1.$$

We say that a password model p is *complete* if and only if it assigns a non-zero probability to any string in Γ.

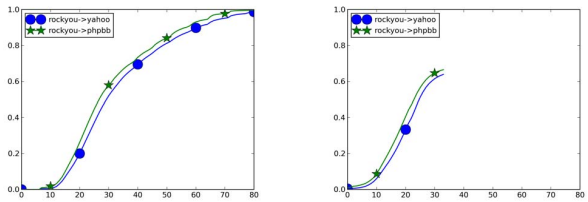
For a password model p to be useful in practice, it should be *efficiently computable*; that is, for any $s \in \Gamma$, computing $p(s)$ takes time proportional to $O(|\Sigma|\cdot length(s))$. For p to be useful in cracking a password, it should be *efficiently enumerable*; that is, for any integer N, it runs in time proportional to $O(N \cdot |\Sigma| \cdot U)$ to output the N passwords that have the highest probabilities according to p. If one desires to make a large number of guesses, then the password generation also needs to be *space efficient*, i.e., the space used for generating N passwords should grow at most sub-linearly in N. Ideally, the space usage (not counting the generated passwords) should be independent from N.

We consider the following three methods/metrics when running a given password model with a testing password dataset. Note that for type-1 research, we fix the model and compare different datasets. For type-2 research, we fix a dataset and compare different models.

Guess-number graphs. Such a graph plots the number of guesses in log scale vs. the percentage of passwords cracked in the dataset. A point (x, y) on a curve means that y percent of passwords are included in the first 2^x guesses (i.e., the 2^x passwords that have the highest probabilities). This approach is logically appealing; however, it is also highly computationally expensive, since it requires generating a very large number of password guesses in decreasing probability. This makes it difficult when we want to compare many different approaches. Furthermore, because of the resource limitation, we are unable to see the effect beyond the guesses we have generated. These motivate us to use the following metrics.

Probability-threshold graphs. Such a graph plots the probability threshold in log scale vs. the percentage of passwords above the threshold. A point (x, y) on a curve means that y percent of passwords in the dataset have probability at least 2^{-x}. To generate such a figure, one needs only to compute the probabilities the model assigns to each password in the testing dataset. For each probability threshold, one counts how many passwords are assigned probabilities above the threshold. For a complete password model, such a curve can show the effect of a password model on the dataset all the way to the point when all passwords in the dataset are included.

Figure 1 below shows a probability-threshold graph and the corresponding guess number graph, for comparing the relative strength of two datasets phpbb and yahoo using a Markov chain model of order 5 trained on the rockyou dataset.



(a) Probability-threshold graph (b) Guess-number graph

Fig. 1: An example

An interesting question is how a probability-threshold graph relates to the corresponding guess-number graph. If one draws conclusions regarding two curves based on the former, do the conclusions hold in the latter?

When comparing the strength of two password datasets using a fixed model, the answer is "yes". The relationship of two guess-number curves (e.g., which curve is to the left of the other, whether and when they cross, etc.) would be exactly the same as that demonstrated by the beginning portions of the corresponding probability-threshold curves; because for a given threshold, exactly the same set of passwords will be attempted. This effect can be observed from Figure 1. The only information missing is that we do not know the exact guess number corresponding to each probability threshold; however, this information is not needed when our goal is to compare two datasets. In addition to being much faster to compute, the probability-threshold graph is able to show the quality of passwords in the set all the way to the point when all passwords are cracked (assuming that the password model assigns a non-zero probability to each allowed password).

When comparing models with a fixed dataset, in general, the answer is no. In the extreme case, consider a model that assigns almost the same yet slightly different probabilities to every possible password in Γ, while using the same ranking as in the testing dataset. As a result, the probabilities assigned to all passwords would all be very small. Such an unrealistic model would perform very well in the guess-number graph; however, the probability-threshold graph would show extremely poor performance, since no password is cracked until a very low probability threshold is reached.

When comparing two password models, whether the conclusions drawn from a probability threshold graph can be carried over to the guess number graph depends on whether the two models have similar rank distributions. Given a password model p, the rank distribution gives for each $i \in \{1, 2, 3, \cdots\}$, the probability of the password that has the i'th highest probability. Thus, if two password models are known to generate very similar rank distributions, then any conclusion drawn from the probability-threshold graph will also hold on the guess number graph, because the same probability threshold will correspond to the same guess number. When two models generate somewhat different rank distributions, then one needs to be careful when interpreting results obtained from probability-threshold graphs. In particular, one may want to also generate the guess number graphs to check whether the results are similar at least for the beginning portion of the probability-threshold graphs. There is the main limitation of using probability-threshold graphs in type-2 research.

Average-Negative-Log-Likelihood (ANLL) and ANLL$_\theta$. Information encoded in a probability-threshold curve can be summarized by a single number that measures the area to the left of the curve, which turns out to be exactly the same as the Average Negative Log Likelihood (ANLL). ANLL has its root in statistical language modeling. A statistical language model assigns a probability to a sentence (i.e., a sequence of words) by means of a probability distribution. Language modeling is used in many natural language processing applications such as speech recognition, machine translation, part-of-speech tagging, parsing and information retrieval. Password modeling can be similarly viewed as trying to approximate as accurately as possible $\mathcal{D}$, an unknown distribution of passwords. We do not know $\mathcal{D}$, and instead have a testing dataset D, which can be viewed as sampled from $\mathcal{D}$. We use p_D to denote the probability distribution given by the dataset D. Representing the testing dataset D as a multi-set $\{s_1, s_2, \ldots, s_n\}$, where a password may appear more than once, the ANLL metric is computed as follows:

$$\mathsf{ANLL}(D|p) = \frac{1}{|D|} \sum_{s \in D} -\log_2 p(s)$$

To see that ANLL equals the area to the left of the probability-threshold curve, observe that when dividing the area into very thin horizontal rows, the length of a row with height between y and $y + dy$ is given by the number of passwords with probability p such that $y \leq -\log_2 p < y + dy$. By summing these areas up, one obtains the ANLL.

ANLL gives the same information as *Empirical Perplexity*, which is the most widely used metric for evaluating statistical language models, and is defined as $2^{\mathsf{ANLL}(D|p)}$.

While using a single number to summarize the information in a probability-threshold curve is attractive, obviously one is going to lose some information. As a result, ANLL has some limitations. First, ANLL is applicable only for complete password models, i.e., those that assign a non-zero probability to each password. In the area interpretation, if a password in the dataset is assigned probability 0, it is never guessed, and the curve never reaches $Y = 1$, resulting in an infinite area. Second, ANLL includes information about cracking 100 percent of the passwords, which may not be what one wants. Since it may be infeasible to generate enough guesses to crack the most difficult passwords, one may care about only the θ portion of passwords that are easy to crack. In that case, one could use ANLL$_\theta$, which we define to be the area to the left of the curve below θ. In this paper, we use ANLL$_{0.8}$ when comparing different models using one single number.

III. Design Space of Password Models

We mostly consider approaches that construct password models without relying on an external dictionary. That is, we consider approaches that can take a training password dataset and learn a password model from it. The performance of approaches that rely on external dictionaries depends very much on the quality of the dictionaries, and in particular, how well the dictionaries match the testing dataset. Such approaches are not broadly applicable, and it is difficult to evaluate their effectiveness.

N-gram models, i.e., Markov chains, are the dominant approach for statistical languages. It is natural to apply them to passwords. A Markov chain of order d, where d is a positive integer, is a process that satisfies

$$P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1) = P(x_i|x_{i-1}, \ldots, x_{i-d})$$

where d is finite and $x_1, x_2, x_3, \ldots$ is a sequence of random variables. A Markov chain with order d corresponds to an n-gram model with $n = d + 1$.

We divide password models into two classes, *whole-string models* and *template-based models*. A template-based model divides a password into several segments, often by grouping consecutive characters of the same category into one segment, and then generates the probability for each segment independently. A whole-string model, on the other hand, does not divide a password into segments.

A. Whole-String Markov Models

Whole-string Markov models are used in John the Ripper (JTR) [3] and Castelluccia et al.'s work on password strength estimation [8]. JTR in Markov mode uses a 1-order Markov chain, in which the probability assigned to a password "$c_1 c_2 \cdots c_\ell$" is

$$P(\text{“}c_1 c_2 \cdots c_\ell\text{”}) = P(c_1|c_0)P(c_2|c_1)P(c_3|c_2)\cdots P(c_\ell|c_{\ell-1}),$$

where $c_0 \notin \Sigma$ denotes a start symbol that is prepended to the beginning of all passwords, and

$$P(c_i|c_{i-1}) = \frac{count(c_{i-1}c_i)}{count(c_{i-1}\cdot)} \quad (1)$$

where $count(c_{i-1}\cdot)$ denotes the number of occurrences of c_{i-1} where it is followed by another character (i.e., where it is not at the end of password), and $count(c_{i-1}c_i)$ gives the number of occurrences of the substring $c_{i-1}c_i$. By definition, we have $count(c_{i-1}\cdot) = \sum_{c_i \in \Sigma} count(c_{i-1}c_i)$. When we use Markov chains of order $d > 1$, we can prepend d copies of the start symbol c_0 to each password.

There are many variants of whole-string Markov models. Below we examine the design space.

Need for Normalization. We note that models used in [3], [8] are not probability models because the values assigned to all strings do not add up to 1. In fact, they add up to $U-L+1$, where U is the largest allowed password length, and L is the smallest allowed password length, because the values assigned to all strings of a fixed length add up to 1. To see this, first observe that

$$\sum_{c_1 \in \Sigma} P(\text{“}c_1\text{”}) = \frac{\sum_{c_1 \in \Sigma} count(c_0c_1)}{count(c_0\cdot)} = 1,$$

and thus the values assigned to all length-1 strings add up to 1. Assume that the values assigned to all length-ℓ strings sum up to 1. We then show that the same is the case for length $\ell + 1$ as follows.

$$\begin{aligned} & \sum_{c_1c_2\cdots c_{\ell+1} \in \Sigma^{\ell+1}} P(\text{“}c_1c_2\cdots c_{\ell+1}\text{”}) \\ = & \sum_{c_1c_2\cdots c_\ell \in \Sigma^\ell} P(\text{“}c_1c_2\cdots c_\ell\text{”}) \times \sum_{c_{\ell+1} \in \Sigma} P(c_{\ell+1}|c_\ell) \\ = & \sum_{c_1c_2\cdots c_\ell \in \Sigma^\ell} P(\text{“}c_1c_2\cdots c_\ell\text{”}) = 1 \end{aligned}$$

The same analysis holds for Markov chains of order $d > 1$. To turn such models into probability models, several normalization approaches can be applied:

Direct normalization. The approach of using the values directly in [3], [8] is equivalent to dividing each value by $(U - L + 1)$. This method, however, more or less assumes that the total probabilities of passwords for each length are the same. This is clearly not the case in practice. From Table II(c), which gives the password length distributions for the datasets we use in this paper, we can see that passwords of lengths between 6 and 10 constitute around 87% of all passwords, and passwords of lengths 11 and 12 add an additional 7%, with the remaining 30 lengths (lengths 4,5,13-40) together contributing slightly less than 6%.

Distribution-based normalization. A natural alternative to direct normalization is to normalize based on the distribution of passwords of different lengths. More specifically, one normalizes by multiplying the value assigned to each password of length m with the following factor:

$$\frac{\text{\# of passwords of length } m \text{ in training}}{\text{total \# of passwords in training}}$$

This approach, however, may result in overfitting, since the training and testing datasets may have different length distributions. From Table II(c), we can see that the CSDN dataset includes 9.78% of passwords of length 11, whereas the PhpBB dataset includes only 2.1%, resulting in a ratio of 4.66 to 1, and this ratio keeps increasing. At length 14, CSDN has 2.41%, while PhpBB has only 0.21%.

End-symbol normalization. We propose another normalization approach, which appends an “end” symbol c_e to each password, both in training and in testing. The probability assigned by an order-1 Markov chain model to a password “$c_1c_2\cdots c_\ell$” is

$$P(c_1|c_0)P(c_2|c_1)P(c_3|c_2)\cdots P(c_\ell|c_{\ell-1})P(c_e|c_\ell),$$

where the probability of $P(c_e|c_\ell)$ is learned from the training dataset. For a string “$c_1c_2\cdots c_U$” of the maximal allowed length, we define $P(c_e|c_U) = 1$.

In this approach, we also consider which substrings are more likely to occur at the end of passwords. In Markov chain models using direct normalization, the prefix of a password is always more likely than the actual password, which may be undesirable. For example, “passwor” would be assigned a higher probability than “password”. The end symbol corrects this situation, since the probability that an end symbol following “passwor” is likely to be significantly lower than the product of the probability that “d” follows “passwor” and that of the end symbol following “password”.

Such a method can ensure that the probabilities assigned to all strings sum up to 1 when the order of the Markov chain is at least as large as L, the smallest allowed password length. To see this, consider the case when $L = 1$. Envision a tree where each string corresponds to a node in the tree, with c_0 being the root node. Each string “$c_0c_1\cdots c_\ell$” has “$c_0c_1\cdots c_{\ell-1}$” as its immediate parent. All the leaf nodes are strings with c_e as the last symbol, i.e., they are actual passwords. Each edge is assigned the transition probability; thus, the values assigned to all edges leaving a parent node add up to 1. Each node in the tree is assigned a value. The root is assigned 1, and every other node is assigned the value of the parent multiplied by the transition probability of the edge from its parent to the node. Thus, the value assigned to each leaf node equals its probability under the model. We note that the value assigned to each node equals the sum of values assigned to its children. It follows that the total probabilities assigned to all leaves add up to be the same as those assigned to the root, which is 1.

When the order of Markov chain, d, is less than L, the minimal required password length, we may have the situation that the total probabilities add up to less than 1, because non-zero probabilities would be assigned to illegal passwords of lengths less than L. The effect is that this method is slightly penalized when evaluated using probability-threshold graphs or ANLLs. We note that when a model wastes half of the total probability of 1 by assigning them to passwords that are too short, the ANLL_θ is increased by θ.

Choice of Markov Chain Order, Smoothing, and Spar-

sity. Another challenging issue is the choice of the order for Markov chains. Intuitively, a higher-order Markov chain enables the use of deeper history information than lower-order ones, and thus may be more accurate. However, at the same time, a higher-order Markov chain runs the risk of overfitting and sparsity, which causes reduced performance. Sparsity means that one is computing transition probabilities from very small count numbers, which may be noisy.

In Markov chain models, especially higher order ones, many calculated conditional probabilities would be 0, causing many strings assigned probability 0. This can be avoided by applying the technique of smoothing, which assigns pseudocounts to unseen strings. Pseudocounts are generally motivated on Bayesian grounds. Intuitively, one should not think that things that one has not yet seen have probability 0. Various smoothing methods have been developed for language modeling. Additive smoothing, also known as Laplace smoothing, adds δ to the count of each substring. When δ is chosen to be 1, this is known as add-one smoothing, which is problematic because the added pseudo counts often overwhelm the true counts. In this paper, we choose δ to be 0.01; the intuition is that when encounting a history "$c_1c_2c_3c_4$", all the unseen next characters, after smoothing come up to slightly less 1, since there are 95 characters.

A more sophisticated smoothing approach is Good-Turing smoothing [13]. This was developed by Alan Turing and his assistant I.J. Good as part of their efforts at Bletchley Park to crack German Enigma machine ciphers during World War II. An example illustrating the effect of Good-Turing smoothing is in [5], where Bonneau calculated the result of applying Good-Turing smoothing to the Rockyou dataset. After smoothing, a password that appears only once has a reduced count of 0.22, a password that appears twice has a count of 0.88, whereas a password that appears $k \geq 3$ times has a reduced count of approximately $k-1$. The "saved" counts from these reduction are assigned to all unseen passwords. Intuitively, estimating the probability of a password that appears only a few times using its actual count will overestimate its true probability, whereas the probability of a password that appears many times can be well approximated by its probability in the dataset. The key observation underlying Good Turing smoothing is that the best estimation for the total probability of items that appear exactly i times is the total probability of items that appear exactly $i+1$ times. In particular, the total probability for all items unseen in a dataset is best estimated by the total probability of items that appear only once.

Grouping. Castelluccia et al. [8] proposed another approach to deal with sparsity. They observed that upper-case letters and special symbols occur rarely, and propose to group them into two new symbols. Here we use Υ for the 26 upper-case letters and Ω for the 33 special symbols. This reduces the size of the alphabet from 95 to 38 (26 lower case letters, 10 digits, and Υ and Ω), thereby reducing sparsity. This approach treats all uppercase letters as exactly the same, and all special symbols as exactly the same. That is, the probability that any upper-case letter follows a prefix is the probability that Υ follows the prefix divided by 26. For example, $P[\text{"aF?b"}]$ for an order 2 model is given by

$$P[\text{a}|s_0s_0]\ \frac{P[\Upsilon|s_0\text{a}]}{26}\ \frac{P[\Omega|\text{a}\Upsilon]}{33}\ P[b|\Upsilon\Omega].$$

We call this the "grouping" method.

We experimented with an adaptation of this method. When encountering an upper-case letter, we assign probabilities in proportion to the probability of the corresponding lower-case letter, based on the intuition that following "swor", "D" is much more likely than Q, just as d is much more like than q. When encountering a special symbol, we use the order-1 history to assign probability proportionally, based on the intuition that as special symbols are rare, a shorter history is likely better. In the adaptation, the probability of the above string will be computed as:

$$P[\text{a}|s_0s_0]\ \frac{P[\Upsilon|s_0\text{a}]P[f|s_0\text{a}]}{\sum_\alpha P[\alpha|s_0\text{a}]}\ \frac{P[\Omega|a\Upsilon]P[?|F]}{\sum_\omega P[\omega|F]}\ P[b|\Upsilon\Omega],$$

where α ranges over all lower-case letters, and ω ranges over all special symbols.

Backoff. Another approach that addresses the issue of order, smoothing, and sparsity together is to use variable order Markov chains. The intuition is that if a history appears frequently, then we would want to use that to estimate the probability of the next character. For example, if the prefix is "passwor", using the whole prefix to estimate the next character would be more accurate than using only "wor". On the other hand, if we observe a prefix that rarely appears, e.g., "!W}ab", using only "ab" as the prefix is likely to be more accurate than using longer history, which likely would assign equal probability to all characters. One way to do this is Katz's backoff model [16]. In this model, one chooses a threshold and stores all substrings whose counts are above the threshold. Let $\pi_{i,j}$ denote the substring of $s_0s_1s_2\cdots s_\ell$ from s_i to s_j. To compute the transition probability from $\pi_{0,\ell-1}$ to $\pi_{0,\ell}$, we look for the smallest i value such that $\pi_{i,\ell-1}$ has a count above the threshold. There are two cases. In case one, $\pi_{i,\ell}$'s count is also above the threshold; thus, the transition probability is simply $\frac{count(\pi_{i,\ell})}{count(\pi_{i,\ell-1})}$. In case two, where $\pi_{i,\ell}$ does not have a count, we have to rely on a shorter history to assign this probability. We first compute b, the probability left after assigning transition probabilities to all characters via case one. We then compute the probabilities for all other characters using the shorter prefix $\pi_{i+1,\ell-1}$, which are computed in a recursive fashion. Finally, we normalize these probabilities so that they add up to b.

Backoff models are significantly slower than other Markov models. The backoff models used in experimentation are approximately 11 times slower than the plain Markov models, both for password generation and for probability estimation.

B. Template-based Models

In the template-based approach, one divides passwords into different templates based on the categories of the characters.

For example, one template is 6 lower-case letters followed by 3 digits, which we denote as $\alpha^6\beta^3$. Such a template has two segments, one consisting of the first 6 letters and the other consisting of last 3 digits. The probability of the password using this template is computed as:

$$P(\text{“passwd123”}) = P(\alpha^6\beta^3)P(\text{“passwd”}|\alpha^6)P(\text{“123”}|\beta^3),$$

where $P(\alpha^6\beta^3)$ gives the probability of the template $\alpha^6\beta^3$, $P(\text{“passwd”}|\alpha^6)$ gives the probability that “passwd” is used to instantiate 6 lowercase letters, and $P(\text{“123”}|\beta^3)$ gives the probability that “123” is used to instantiate 3 digits. To define a model, one needs to specify how to assign probabilities to templates, and to the different possible instantiations of a given segment. Template-based models are used in [21] and the PCFG$_W$ model [25].

Assigning probabilities to templates. In [21], the probabilities for templates are manually chosen because this work was done before the availability of large password datasets. In [25], the probability is learned from the training dataset by counting. That is, the probability assigned to each template is simply the number of passwords using the template divided by the total number of passwords. Using this approach means that passwords that use a template that does not occur in the training dataset will have probability 0.

We consider the alternative of applying Markov models to assign probabilities to templates. That is, we convert passwords to strings over an alphabet of $\{\alpha, \beta, \upsilon, \omega\}$, representing lower-case letters, digits, upper-case letters, and symbols respectively. We then learn a Markov model over this alphabet, and assign probabilities to a template using Markov models.

Assigning probabilities to segments. In [21], the probabilities of letter segments are assigned using a Markov model learned over natural language, and the probabilities for digit segments and symbol segments are assigned by assuming that all possibilities are equally likely. In the PCFG$_W$ model [25], the probabilities for digit segments and symbol segments are assigned using counting from the training dataset, and letter segments are handled using a dictionary. That is, for an α^6 segment, all length-6 words in the dictionary are assigned an equal probability, and any other letter sequence is assigned probability 0. For a β^3 template, the probability of “123” is the number of times “123” occurs as a 3-digit segment in the training dataset divided by the number of 3-digit segments in the training dataset.

We consider template models in which we assign segment instantiation probabilities via both counting and Markov models. Table I summarizes the design space of password models we consider in this paper.

PCFG$_W$ model does not fit in the models in Table I, as it requires as input a dictionary in addition to a training dataset. In this paper, we considered 3 instantiations of PCFG$_W$: the first uses the dictionary used in [25]; the second uses the OpenWall dictionary; and the third generates the dictionary from the training set. We note that the last instantiation is essentially the template-based model using counting both for assigning template probabilities and for instantiating segments.

C. Password Generation

To use a password model for cracking, one needs to be able to generate passwords in decreasing probability.

In whole-string Markov-based methods, the password search space can be modeled as a search tree. As described earlier, the root node represents the empty string (beginning of the password), and every other node represents a string. Each node is labeled with the probability of the string it represents.

One algorithm for password guess generation involves storing nodes in a priority queue. The queue, arranged in order of node probabilities, initially contains the root node of the search tree. We then iterate through the queue, popping the node with the greatest likelihood at each iteration. If this is an end node (any non-root node for uniform or length-based normalization, or nodes with end symbol for end-symbol normalization), we output the string as a password guess. Otherwise, we calculate the transition probability of each character in the character set and add to the priority queue a new child node with that character and the associated probability. This algorithm is guaranteed to return password guesses in decreasing order of probability (as estimated by the model), due to the iteration order and property that each node's probability is smaller than that of its parent. The algorithm terminates once the desired number of guesses has been generated.

The priority-queue method, however, is not memory efficient, and does not scale for generating a large number (e.g., over 50 million) of guesses. Since each node popped from the queue can result in multiple nodes added to the queue, the queue size is typically several times of the number of guesses generated. To reduce the memory footprint, we propose a threshold search algorithm.

The threshold search algorithm, similar to the iterative deepening state space search method, conducts multiple depth-first traversals of the search tree. In the i'th iteration, it generates passwords with probabilities in a range $(\rho_i, \tau_i]$, by performing a depth-first traversal of the tree, pruning all nodes with probability less than ρ_i, and outputting all passwords with probability in the target range. To generate n password guesses, we start with a conservative range of $(\rho_1 = \frac{1}{n}, \tau_1 = 1]$. After the i'th iteration, if we have generated $m < n$ passwords so far, we start another iteration with $(\rho_{i+1} = \frac{\rho_i}{\max(2, 1.5n/m)}, \tau_{i+1} = \rho_i]$. That is, when $m < 0.75n$, we halve the probability threshold. We have observed empirically that halving ρ result in close to twice as many passwords being generated. We may overshoot and generally more than n passwords, but are very unlikely to generate over $2n$ guesses. The average runtime complexity of this algorithm as $O(n)$, or linear on n. The memory complexity (not including the model data or generated passwords) is essentially constant, as the only data structure needed is a stack of capacity $U+1$. We use this framework to efficiently generate Markov model-based guesses in the experiments. Slight adjustments need to be made for distribution-based normalization. This method, however,

Whole-string Markov models	normalization methods: (1) direct, (2) distribution-based, (3) end symbol order of Markov chain: $1, 2, 3, 4, \cdots$ or variable (backoff) dealing with sparsity: (1) plain, (2) grouping (3) adapted grouping smoothing methods: (1) no smoothing, (2) add-δ smoothing, (3) Good-Turing smoothing
Template-based models	template probability assignment: (1) Counting, (2) Markov model segment probability assignment: (1) Counting, (2) Markov model

TABLE I: Design space for password probability models

does not apply to template-based models. Through probability-threshold graphs, we have found that unless Markov models are used to instantiate templates, template-based models perform rather poorly. However, when Markov models are used, efficient generation for very large numbers of passwords appears quite difficult, as one cannot conduct depth-first search and needs to maintain a large amount of information for each segment. In this paper, we do not do password generation for template-based models other than the PCFG_W model.

IV. Experimental Methodologies

In this section, we describe our experimental evaluation methodologies, including the dataset we use and the choice of training/testing scenarios.

Datasets. We use the following six password datasets downloaded from public Web sites. We use only the password information in these datasets and ignored all other information (such as user names and/or email addresses included in some datasets). The **RockYou** dataset [1] contains over 32 million passwords leaked from the social application site Rockyou in December 2009. The **PhpBB** dataset [1] includes about 250K passwords cracked by Brandon Enright from MD5 hashes leaked from Phpbb.com in January 2009. The **Yahoo** dataset includes around 450K passwords published by the hacker group named D33Ds in July 2012. The **CSDN** dataset includes about 6 million user accounts for the Chinese Software Developer Network (CSDN), a popular IT portal and a community for software developers in China [2]. The **Duduniu** dataset includes about 16 million accounts for a paid online gaming site in China. The **178** datasets includes about 10 million passwords for another gaming website in China. All 3 chinese dataset were leaked in December 2011.

The first 3 datasets are from American websites, and the last 3 are from Chinese websites. The 6 datasets together have approximately 60 million passwords. Information about these datasets is presented in Table II.

Data cleansing. We performed the following data cleansing operations. First, we removed any password that includes characters beyond the 95 printable ASCII symbols. This step removed 0.034% of all passwords. In the second step, we removed passwords whose length were less than 4 or greater than 40. As almost any system that uses passwords will have a minimal length requirement and a maximum length restriction, we believe that such processing is reasonable. In total we removed 0.037% of passwords that are too short, and 0.002% of passwords that are too long. Length-4 passwords account for 0.225% of the datasets, and we chose not to remove them. Table II(a) gives detailed information on cleansing.

Dataset statistics. Table II(b) shows the percentages of passwords that appeared $1, 2, 3, 4, 5+$ times in the datasets. Recall that the Good-Turing method estimates that the total probabilities of unseen passwords to be that of unique passwords. We see that the two smallest datasets, PhpBB and Yahoo, have significantly higher percentages of unique passwords, 64.16% and 69.83% respectively, compared to 36.43% for Rockyou. When combining all 6 datasets, approximately 40% are unique.

Table II(c) shows the length distributions of the password datasets, showing the most common password lengths are 6 to 10, which account for 87% of the whole dataset. One interesting aspect is that the CSDN dateset has much fewer length 6 and 7 passwords than other datasets. One explanation is that the website started enforcing a minimal length-8 password policy early on, and only users who have accounts near the beginning have shorter passwords.

Table II(d) shows the character distribution. It is interesting to note that passwords in American datasets consist of about 27% digits and 69% lower-case letters, while those in Chinese datasets are the opposite, with 68% digits and 30% lower-case letters. This is likely due to both the fact that the Roman alphabet is not native to Chinese, and the fact that digit sequences are easier to remember in Chinese. For each digit, there are many chinese characters that have similar sounds, making it easy to find digit sequences that sound similar to some easy-to-remember phrases. Indeed, we found many such sequences occurring frequently in Chinese datasets.[1]

Table II(e) shows the frequencies of different patterns. While all lower-case passwords are the most common in American datasets (41.65%), they account for only 8.93% of Chinese datasets. The overall most common pattern is lowercase followed by digits (33.02%), due to the fact that this is the most common in Chinese datasets (40.05%). This is followed by all lower-case, all digits, and digits followed by lower-case; these top-4 patterns account for close to 90% of all passwords. Upper-case letters are most commonly seen preceding a digit sequence, or in all-uppercase passwords. We also note that the pattern distribution shows a lot of variation across different datasets.

Training/Testing Scenarios. We now describe our selection of the training and testing datasets. We decided against merg-

[1]One such sequence is "5201314", which sounds similar to a phrase that roughly means "I love you forever and ever". Both "520" and "1314" are extreme frequent in Chinese datasets.

TABLE II: Statistics information of the datasets

(a) Data Cleaning of the datasets

Dataset	Size after cleansing		Removed						
	Unique	Total	non-ascii Unique	non-ascii Total	too-short Unique	too-short Total	too-long Unique	too-long Total	Percentage of All Removed
RockYou	14325681	32575584	14541	18038	2868	7914	1290	1346	0.08%
PhpBB	183400	253512	45	45	944	1864	0	0	0.75%
Yahoo	342197	442288	0	0	283	497	0	0	0.11%
CSDN	4037139	6427522	314	355	465	754	0	0	0.02%
Duduniu	6930996	10978339	1485	1979	3685	10881	28	28	0.12%
178	3462280	9072960	0	0	3	5	1	1	0.00%

(b) Password count & frequency information

	All	All American	RockYou	PhpBB	Yahoo	All Chinese	CSDN	Duduniu	178
all	59750205	33271384	32575584	253512	442288	26478821	6427522	10978339	9072960
Pecentage due to Unique	40.92%	37.09%	36.43%	64.16%	69.83%	45.74%	55.72%	51.83%	31.30%
Pecentage due to Twice	9.32%	8.15%	8.12%	9.79%	9.38%	10.73%	10.79%	14.20%	7.80%
Pecentage due to 3 Times	4.02%	3.58%	3.59%	3.86%	3.60%	4.46%	4.55%	5.96%	3.65%
Pecentage due to 4 Times	2.36%	2.25%	2.25%	2.31%	1.94%	2.47%	2.51%	3.18%	2.19%
Pecentage due to 5+ Times	43.38%	48.93%	49.61%	19.88%	15.25%	36.41%	29.84%	24.83%	55.07%

(c) Password length information

	All	All American	RockYou	PhpBB	Yahoo	All Chinese	CSDN	Duduniu	178
4	0.2253%	0.2444%	0.2164%	3.1833%	0.6215%	0.2013%	0.1041%	0.4226%	0.0023%
5	2.4012%	4.0466%	4.0722%	5.7110%	1.2028%	0.3338%	0.5142%	0.5035%	0.0006%
6	19.4561%	25.9586%	26.0553%	27.4212%	17.9994%	11.2856%	1.2954%	9.1683%	20.9249%
7	16.7693%	19.2259%	19.2966%	17.8169%	14.8313%	13.6824%	0.2696%	16.3914%	19.9063%
8	22.6479%	20.1374%	19.9886%	27.3967%	26.9336%	25.8024%	36.3793%	23.0194%	21.6768%
9	16.5047%	12.1343%	12.1198%	9.1562%	14.9125%	21.9962%	24.1479%	24.6739%	17.2318%
10	11.7203%	9.0805%	9.0650%	5.3276%	12.3795%	15.0372%	14.4810%	20.0710%	9.3403%
11	4.7320%	3.5711%	3.5659%	2.0985%	4.7976%	6.1907%	9.7820%	3.4728%	6.9350%
12	2.3497%	2.1347%	2.1053%	1.0611%	4.9124%	2.6199%	5.7475%	1.1962%	2.1269%
13	1.2445%	1.3008%	1.3170%	0.4307%	0.6005%	1.1738%	2.6106%	0.5061%	0.9638%
14	0.8673%	0.8490%	0.8609%	0.2142%	0.3373%	0.8902%	2.4105%	0.3003%	0.5269%
15	0.5062%	0.5431%	0.5515%	0.0935%	0.1890%	0.4598%	1.1719%	0.1895%	0.2822%
16	0.3128%	0.3870%	0.3931%	0.0505%	0.1286%	0.2195%	0.7716%	0.0255%	0.0633%
17	0.0815%	0.1208%	0.1225%	0.0142%	0.0592%	0.0320%	0.1090%	0.0124%	0.0013%
18	0.0550%	0.0759%	0.0770%	0.0110%	0.0283%	0.0287%	0.0918%	0.0107%	0.0058%
19	0.0322%	0.0486%	0.0494%	0.0036%	0.0199%	0.0115%	0.0356%	0.0060%	0.0013%
20	0.0331%	0.0412%	0.0415%	0.0032%	0.0400%	0.0229%	0.0782%	0.0082%	0.0015%
21-30	0.0575%	0.0947%	0.0964%	0.0056%	0.0067%	0.0110%	0.0000%	0.0191%	0.0090%
31-40	0.0034%	0.0053%	0.0055%	0.0012%	0.0000%	0.0012%	0.0000%	0.0029%	0.0000%

(d) Password characters information

	All	All American	RockYou	PhpBB	Yahoo	All Chinese	CSDN	Duduniu	178
all	490102135	262266576	256756616	1857611	3652349	227835559	60788099	93174301	73873159
digit	46.20%	27.28%	27.35%	23.12%	24.56%	67.99%	67.41%	64.74%	72.55%
lower	51.06%	68.87%	68.78%	73.65%	72.55%	30.55%	30.06%	33.79%	26.87%
special	0.48%	0.67%	0.68%	0.32%	0.49%	0.25%	0.62%	0.12%	0.13%
upper	2.26%	3.18%	3.19%	2.91%	2.39%	1.21%	1.91%	1.35%	0.46%

(e) Password pattern information: **L** denotes a lower-case sequence, **D** for digit sequence, **U** for upper-case sequence, and **S** for symbol sequence; patterns differ from templates in that patterns do not record length of sequence

	ALL		Percentage of the patterns in American datasets				Percentage of the patterns in Chinese datasets			
	Percentage of the pattern	the most popular string	All American	rockyou	phpbb	yahoo	All Chinese	csdn	duduniu	178
LD	33.22%	a123456	27.79%	27.71%	19.26%	38.31%	40.05%	26.15%	55.57%	31.12%
L	27.15%	password	41.65%	41.70%	50.08%	33.05%	8.93%	11.65%	7.29%	9.00%
D	24.50%	123456	15.77%	15.94%	11.94%	5.86%	35.48%	45.02%	19.48%	48.07%
DL	4.81%	123456aa	2.57%	2.54%	2.05%	5.32%	7.63%	5.89%	9.79%	6.25%
LDL	1.60%	love4ever	1.66%	1.62%	3.66%	3.31%	1.53%	1.64%	1.68%	1.27%
UD	1.48%	A123456	1.33%	1.35%	0.37%	0.56%	1.67%	1.62%	2.57%	0.62%
U	0.94%	PASSWORD	1.48%	1.50%	0.73%	0.40%	0.26%	0.47%	0.21%	0.15%
ULD	0.64%	Password1	0.96%	0.94%	1.04%	2.48%	0.24%	0.50%	0.26%	0.05%
DLD	0.43%	123aa123	0.43%	0.42%	0.79%	0.94%	0.44%	0.52%	0.44%	0.38%
LDLD	0.42%	hi5hi5	0.43%	0.42%	1.03%	0.97%	0.40%	0.47%	0.31%	0.47%
UL	0.40%	Password	0.65%	0.65%	1.22%	0.70%	0.09%	0.09%	0.15%	0.01%
LSD	0.40%	xxx_01	0.66%	0.66%	0.33%	0.17%	0.27%	0.21%	0.05%	0.01%
LSL	0.40%	rock you	0.50%	0.50%	0.17%	0.39%	0.08%	0.66%	0.21%	0.08%
LS	0.38%	iloveyou!	0.64%	0.65%	0.16%	0.20%	0.07%	0.14%	0.05%	0.04%
DU	0.23%	123456A	0.15%	0.15%	0.11%	0.06%	0.34%	0.46%	0.46%	0.11%

TABLE III: Six experimental scenarios.

#	name	Training	Testing
1	Rock→Ya+Ph	Rockyou	Yahoo+PhpBB
2	Du+178→CSDN	Duduniu+178	CSDN
3	CS+178→Dudu	CSDN+178	Duduniu
4	CS+Du→178	CSDN+Duduniu	178
5	Chin→Ya+Ph	Three Chinese	Yahoo+PhpBB
6	Rock→CSDN	Rockyou	CSDN

ing all datasets into one big dataset and then partitioning it into training and testing, since we feel that represents unrealistic scenarios in practice. For example, this causes similar length, character, and pattern distributions in training and testing; furthermore, any frequent password tends to appear both in training and testing. Instead, in each scenario, we chose some of the datasets for training, and another for testing. Table III lists the 6 scenarios we use in this paper. Scenarios 1-4 have training and testing from within the same group (American or Chinese). We merge Yahoo and PhpBB together because they are both small (containing less than one million passwords when combined) when compared with other datasets (all contain more than 6 million). Scenarios 2-4 resemble cross-validation, rotating among the 3 Chinese datasets, each time training with 2 and testing with the remaining 1. Scenario 5 trains on all Chinese datasets and test on the American datasets Yahoo+PhpBB. Scenario 6 trains on Rockyou and tests on the Chinese dataset CSDN. By comparing scenario 5 against 1, and scenario 6 against 2, one can observe the effect of training on a mismatched dataset. We present detailed results using graphs for scenario 1 and 2, and only $\text{ANLL}_{0.8}$ for other scenarios, because of space limitation.

V. Experimental Results

Experimental results are presented using guess-number graphs, probability-threshold graphs, and $\text{ANLL}_{0.8}$ values. The algorithm naming convention is as follows. Names for whole-string models start with "ws". For example, ws-mc-b_{10} is Markov chain with backoff and frequency threshold 10, and ws-mc_i is order-i Markov chain with add-δ smoothing for $\delta = 0.01$. The postfix -g is for grouping, -ag for grouping after our adaption, -gts for Good-Turing smoothing, and -end, -dir, and -dis are for the three normalization approaches. Names for template-based models start with "tb"; for example, tb-co-mc_i is the template-based model using the counting-based method for assigning probabilities to templates and an order-i Markov chain model for segment instantiation. We note that using this notation, tb-co-co is PCFG_W with dictionary generated from the training dataset.

The Figures. Fig 2 gives 8 graphs for Scenario 1. Fig 2(a) shows the rank vs. probability based on generated passwords. One can see that for the three Markov models shown there, one can translate log of rank into negative log of probability via an additive factor in the range of 3 to 8. That is, a password that has probability $\frac{1}{2^y}$ is likely to have a rank of around 2^{y-a}, which a is mostly between 3 and 8, and seems to gets close to around 6 as x increases. We conjecture that this trend will further hold as x increases, and the gap between different curves would shrink as x increases. One support is that these curves have to cross at some point due to the fact that the probabilities for each model add up to 1. Analyzing such Markov models to either prove or disapprove this conjecture is interesting future research, since it affects to what extend one can use probability threshold graphs instead of guess number graphs to compare these models with each other.

Fig 2(b) shows the guess-number graph. We include results from ws-mc_5-end, three instantiations of PCFG_W (using dic-0294, the dictionary used in [25], using the Openwall dictionary [4], and using a dictionary generated from the training dataset), and JTR in three different modes (incremental brute-force, markov chain, and wordlist/dictionary mode with openwall dictionary). We can see that the three PCFG_W and the three JTR methods clearly underperform the Markov model. We note that jtr-mc1 seems to pick up rather quickly, which matches the shape of ws-mc1 in 2(f). Another observation is PCFG_W with training dataset as dictionary (i.e., tb-co-co) outperforms the other two instantiations.

Fig 2(c) plots both guess-number curves and probability threshold curves for 3 password models on the same graph, and one can see that the guess-number curve for any model approximately matches the the probability threshold curve if one shifts them to the right.

Fig 2(d) shows the probability threshold curves for all template-based models and one whole-string model, namely ws-mc-b_{10}-end. Here, Laplace smoothing with $\delta = 0.01$ is applied to markov chains in all templeate models. PCFG_W with dic-0294 performs the worst, and PCFG_W with Openwall dictionary performs slightly better. Compared with other models, these two cover the least passwords at any probability threshold. With probability threshold at 2^{-80}, the former covers slightly over 50% of all passwords, and the latter covers close to 60%. The two curves that both use counting to instantiate segments (tb-co-co and tb-mc_5-co) almost overlap; they perform better than PCFG_W with external dictionaries. On lower probability thresholds, they, together with ws-mc-b_{10}, are the best-performing methods At threshold 2^{-80}, they cover around 75% of passwords. This suggests that learning from the dataset is better than using existing dictionaries in PCFG_W. When we replace counting with Markov models for instantiating segments, we see another significant improvement at higher probability threshold. The model tb-co-mc_5 covers more than 90% of passwords (at threshold 2^{-80}), and tb-mc_5-mc_5, which takes advantage of smoothing, covers close to 100% passwords. This improvement, however, comes at the cost of slightly worse performance than tb-co-co and tb-mc_5-co at lower probability thresholds. In other words, whether to use counting or Markov chains to generate probabilities for templates shows a small difference. Using counting to instantiate segments shows an overfitting effect, performing well at low thresholds, but worse at higher ones. Whole-string Markov with backup (ws-mc-b_{10}-end) almost always has the best performance at any threshold.

Fig 2(e) compares the effect of no smoothing, add-δ smooth-

ing, and Good-Turing smoothing on Markov models of order 4 and order 5. When $x < 35$, smoothing makes almost no difference, one simply sees that order 5 outperforms order 4. This is to be expected, since the smoothing counts make a difference only for strings of low probabilities. For larger x values, however, smoothing makes a significant difference, and the difference is much more pronounced for order 5 than for order 4. The order 5 model without smoothing performs the worst, due to overfitting. Good-Turing smoothing underperforms add-δ smoothing, and results in significant overfitting for order 5.

Fig 2(f) compares the effect of different orders in Markov chain models. We see that higher-order chains perform better for smaller x values, but are surpassed by lower-order chains later; however, backoff seems to perform well across all ranges of x.

Fig 2(g) demonstrates the effect of normalization. Direct normalization performs the worst, while distribution based normalization performs slightly better than end-symbol normalization.

As can be seen from Table IIc, Yahoo+PhpBB have between 40% and 45% passwords that are of length less than 8. Since many modern websites require passwords to be at least 8 characters long, one may question to what extent results from the above figures are applicable. To answer this question, we repeat figure Fig 2(d) by using only passwords that are at least 8 characters for evaluation. The result is shown in Fig 2(h). Note that while all curves are somewhat lower than the corresponding ones in Fig 2(d); they tell essentially the same story.

Fig 3 gives the same 8 graphs for scenario 2, which use Chinese datasets for training and testing. The observations made above similarly apply. One minor difference is that in Fig 2(d), the performance of PCFG with external dictionaries are worse than in Scenario 1. Since the Chinese datasets consist of more passwords that use only digit sequences, and thus are intuitively weaker, this may seem a bit counter-intuitive. This is because PCFG$_W$ uses only digit sequences that appear in the training dataset to instantiate password guesses, and thus does perform well when lots of digits are used. When Markov chains with smoothing are used to instantiate the segments, one obtains a more significant improvement than in Scenario 1.

Fig 4 shows two of these graphs for each of the other 4 scenarios. These two are the graph (d) for a probability threshold graph and graph (b) for a guess number graph. They mostly give the same observations. We note that in Fig 4(f), we see that PCFG$_W$ with dictionary from the training dataset starts out-performing ws-mc$_5$-end. Note that in scenario, we train on the Chinese dataset and the use American datasets to evaluate, thus, a higher-order Markov chain does not perform very well. From Fig 4(e), however, we can see that the variable-order ws-mc-b$_{10}$ remains the best-performing method.

ANLL Table. ANLL$_{0.8}$ values for all six scenarios are given in Table IV. Using this format, we can compare more models directly against each other, with the limitation that these results need to be interpreted carefully. Some models assign probability 0 to some passwords; their ANLLs are not well-defined and thus not included. Results for those models are presented using graphs. Because of space limitation, we exclude some ANLL data for some other combinations (such as grouping with Good Turing smoothing or template-based with two different orders).

Many observations can be made from Table IV. First, backoff with end-symbol normalization is gives the best result overall, as it produces results that are among the best across all scenarios. Especially, for Scenario 1, which we consider to be the most important one, it produces the best overall result. Several other models perform quite close. It seems that using a Markoc-chain of an order that is high enough, but not too high, and with some ways to deal with overfitting, would perform reasonably well.

Second, for most other models, distribution-based normalization performs the best, followed by end-symbol normalization. Direct normalization, which was implicitly used in the literature, performs the worst. Yet, for backoff, end-symbol normalization performs the best. There seems to exist some synergy between backoff and end-symbol normalization. One possible reason is that as backoff uses variable-length Markov chains, it can recognize long common postfixes of passwords so that it can end a password appropriately, instead of depending only on the length of passwords for normalization. With fixed-length Markov chains, one does not have this benefit.

Third, on the effect of smoothing, Good-Turing smoothing performs unexpectedly poorly, especially for higher-order Markov chains; it seems that they tend to cause more overfitting, a phenomenon also shown in Figure 2(e) and 3(e). For higher orders Markov models, add-δ smoothing, grouping, adapted grouping, and template-based models all perform similarly; they are just slightly worse than backoff with end-symbol normalization.

Fourth, for most models, the Markov chain order that gives the best results varies from scenario to scenario. For Scenario 1, order-5 appears to be the best. Yet for the scenarios with Chinese datasets (2, 3, 4), order 3 and 4 generally outperform order 5. One obvious reason is the slightly smaller training dataset. Also, because the Chinese datasets use digits much more than letters, they contain even non-digit sequences for training, resulting in better performance for lower-order Markoc chains. Again, this demonstrates the benefit of using a variable-order model like backoff, since one does not need to choose the appropriate order.

Fifth, comparing the pair of scenarios 5 and 1, and the pair of 6 and 2, one can see a difference in ANLL$_{0.8}$ of about 2 to 4 in each case; this demonstrates the importance of training on a similar dataset.

Sixth, comparing scenarios 2, 3, and 4, we can see that 178 is clearly the weakest password dataset among the 3, which is corroborated by evidence from Table II. In 178, 55% of passwords are those appearing more than 5 times (compared to 30% and 25% for CSDN and Duduniu); close to 21% are length 6 (compared to 1.3% and 9%); 48% are all digits

TABLE IV: $\text{ANLL}_{0.8}$ results; end, dir, and dis stand for end-symbol, direct, and distribution-based normalization, respectively. We use boldface to highlight the best results within each of the first 4 scenarios. The last 2 scenarios represent mismatches between training and testing.

	1: Rock→Ya+Ph			2: Du+178→CSDN			3: CS+178→Dudu			4: CS+Du→178			5: Chin→Ya+Ph			6: Rock→CSDN		
Algorithm	end	dir	dis	end	dir	dis	end	dir	dis	end	dir	dis	end	dir	dis	end	dir	dis
ws-mc-b_{10}	**22.9**	25.5	23.6	**19.6**	21.3	20.0	**21.5**	23.2	21.6	**12.9**	14.2	13.2	27.0	28.8	27.1	22.5	23.9	22.9
ws-mc-b_{25}	**23.3**	25.8	23.9	**19.7**	21.4	20.1	21.6	23.4	21.7	**12.9**	14.2	13.2	27.6	29.3	27.5	22.6	24.1	23.0
ws-mc_1	28.4	29.3	27.3	22.2	22.6	21.3	24.0	24.5	22.8	14.9	15.3	14.1	31.1	31.9	30.0	24.1	24.2	23.0
ws-mc_2	26.9	28.0	26.1	20.9	21.5	20.2	22.8	23.5	21.9	13.7	14.5	13.3	30.2	31.1	29.2	23.3	23.6	22.5
ws-mc_3	25.2	26.6	24.8	20.1	20.9	**19.6**	22.0	22.9	**21.3**	13.2	13.9	**12.9**	28.5	29.6	27.8	22.8	23.2	22.1
ws-mc_4	23.9	25.6	23.7	19.9	20.8	**19.5**	21.7	22.8	**21.1**	**12.9**	13.8	**12.8**	28.1	29.3	27.6	22.9	23.4	22.4
ws-mc_5	23.5	25.3	23.5	20.4	21.3	20.1	22.4	23.4	21.8	**12.9**	13.8	**12.8**	29.2	30.1	28.3	23.9	24.4	23.4
ws-mc_1-g	28.4	29.4	27.4	22.3	22.8	21.4	24.0	24.5	22.8	14.9	15.3	14.1	31.1	31.9	30.0	24.1	24.2	23.1
ws-mc_2-g	27.0	28.2	26.2	21.1	21.6	20.3	22.9	23.6	21.9	13.8	14.5	13.4	30.2	31.1	29.3	23.4	23.7	22.6
ws-mc_3-g	25.3	26.8	24.9	20.3	21.0	**19.7**	22.1	23.0	**21.4**	13.3	14.0	**12.9**	28.6	29.6	27.8	22.9	23.3	22.2
ws-mc_4-g	24.0	25.7	23.9	20.0	20.9	**19.6**	21.8	22.8	**21.2**	13.0	13.8	**12.8**	28.0	29.3	27.5	22.9	23.4	22.4
ws-mc_5-g	23.5	25.3	**23.4**	20.3	21.3	20.0	22.4	23.4	21.8	13.0	13.9	**12.9**	28.5	29.6	27.8	23.6	24.2	23.1
ws-mc_6-g	24.0	25.6	23.7	22.3	23.2	21.9	23.9	24.8	23.1	13.4	14.3	13.3	29.1	29.9	28.1	26.2	26.7	25.5
ws-mc_1-ag	28.3	29.3	27.3	22.2	22.6	21.2	24.0	24.5	22.8	14.9	15.3	14.1	31.1	31.8	29.9	24.0	24.1	23.0
ws-mc_2-ag	26.9	28.0	26.1	21.0	21.5	20.2	22.8	23.6	21.9	13.8	14.5	13.3	30.2	31.1	29.3	23.3	23.6	22.4
ws-mc_3-ag	25.2	26.7	24.8	20.2	20.9	**19.6**	22.1	23.0	**21.3**	13.2	13.9	**12.9**	28.5	29.6	27.8	22.8	23.2	22.1
ws-mc_4-ag	23.9	25.6	23.7	19.9	20.8	**19.5**	21.7	22.8	**21.2**	**12.9**	13.8	**12.8**	28.0	29.2	27.5	22.8	23.3	22.2
ws-mc_5-ag	**23.3**	25.2	**23.3**	20.2	21.1	19.9	22.3	23.4	21.8	**12.9**	13.8	**12.8**	28.5	29.5	27.8	23.5	24.1	23.0
ws-mc_6-ag	23.8	25.5	23.6	22.1	23.0	21.8	23.8	24.7	23.1	13.4	14.3	13.3	29.1	29.9	28.1	26.1	26.5	25.4
ws-mc_1-gts	28.4	29.3	27.3	22.2	22.6	21.3	24.0	24.5	22.8	14.9	15.3	14.1	31.1	31.9	30.0	24.1	24.2	23.0
ws-mc_2-gts	26.9	28.0	26.1	20.9	21.5	20.2	22.8	23.5	21.9	13.7	14.5	13.3	30.2	31.1	29.2	23.3	23.6	22.5
ws-mc_3-gts	25.2	26.7	24.8	20.2	20.9	**19.6**	22.0	22.9	**21.3**	13.2	13.9	**12.9**	28.6	29.6	27.8	22.8	23.2	22.1
ws-mc_4-gts	24.2	25.9	24.1	20.1	20.9	**19.7**	21.9	23.0	**21.4**	13.0	13.8	**12.8**	29.3	30.5	28.7	23.4	23.8	22.8
ws-mc_5-gts	25.4	26.9	25.1	21.6	22.4	21.2	24.5	25.2	23.7	13.3	14.2	13.2	33.8	34.1	32.3	26.5	26.8	25.7
tb-mc_1-mc_1	28.5	29.4	27.4	22.3	22.7	21.3	23.9	24.5	22.8	14.9	15.3	14.1	31.1	31.9	30.0	24.0	24.2	23.0
tb-mc_2-mc_2	27.2	28.3	26.3	21.3	21.8	20.4	23.1	23.8	22.1	14.3	14.8	13.7	30.3	31.2	29.3	23.4	23.7	22.6
tb-mc_3-mc_3	25.8	27.1	25.2	20.6	21.2	19.9	22.3	23.2	21.6	13.6	14.2	13.1	28.8	29.8	28.0	22.9	23.3	22.2
tb-mc_4-mc_4	24.6	26.1	24.2	20.3	21.0	**19.7**	22.0	23.0	**21.3**	13.2	14.0	**12.9**	27.9	29.0	27.2	22.6	23.1	22.0
tb-mc_5-mc_5	23.9	25.6	23.7	20.1	21.0	**19.7**	21.7	22.9	**21.2**	13.0	13.8	**12.8**	27.6	28.9	27.1	22.6	23.3	22.1

(compared to 45% for CSDN and 19.5% for Duduniu; recall that passwords in CSDN tend to be significantly longer).

VI. Related Work

One active research area in recent years is to study the quality of users' password choices under different scenarios, e.g., when facing different password policies [18], [17], when presented with different password strength meters [23], [10], when forced to change passwords due to organizational change of password policies [22], when forced to change passwords due to expiration [26], and when "persuaded" to include extra randomness in their password choices [12]. In a similar study, Mazurek et al. [20] collected over 25,000 real passwords from CMU and studied passwords from different groups of users.

In this area, earlier work uses a combination of standard password cracking tools such as John the Ripper (JTR) [3] and ad hoc approaches for estimating the information entropy of a set of passwords. One such approach is NIST's recommended scheme for estimating the entropy of one password, which is mainly based on their length [7]. Florencio and Herley [11], Forget et al. [12], and Egelman et al. [10] all use the formula where the bit strength of a password is considered to be $\log_2((\text{alpha. size})^{len})$; for example, a 9-character password that contains both upper and lower case characters and digits is considered to have bit strength of $\log_2(26+26+10)^9 \approx 53.59$. This clearly overestimates the strength of passwords. A more sophisticated approach, which considers how many numbers, symbols, uppercase are used and where they appear when computing the entropy, was developed in [22].

Weir et al. developed the PCFG_W model [25] and argued that entropy estimation methods such as that recommended by NIST are inaccurate [24]. Instead, they propose to use the guess number of passwords. The combination of the PCFG_W model or its variants and guess numbers becomes the standard method in password security research in recent years [17], [18], [15], [23], [20]. Our research suggests that probability-threshold graphs are a better tool than guess number graphs, and Markov chain based models are likely to offer better choices than the PCFG_W model.

John the Ripper [3], one of the most popular password cracking tools, has several modes for generating password guesses. The wordlist mode uses a dictionary plus various mangling rules. The incremental mode uses trigraph frequencies, i.e., frequencies for the triple of character, position, and password length. The Markov mode uses Markov chains of order 1. Narayanan and Shmatikov [21] proposed a template-based model, with Markov chain being used for assigning probability to letter-based segments. Experimental results show that these approaches all significantly underperform higher-order or variable-order Markov models.

Castelluccia et al. [8] proposed to use whole-string Markov models for evaluating password strengths. Our work differs in that we evaluate the performance of this model and compare

it with a large number of other models in a design space using our methodology. Dell'Amico et al. [9] compared the cracking effectiveness of Markov models on the "Italian", "Finnish", and "MySpace" datasets by computing the estimated probabilities for each password in the datasets and using an approximation algorithm to compute the number of guesses for each threshold. They did not consider normalization, smoothing, etc. Furthermore, their results show very small search spaces for higher Markov models, which seems to suggest that their approximation algorithm underestimates the search space.

Malone and Maher [19] investigated how well password distributions fit the zipf distribution. They fitted the Rockyou dataset to a zipf distribution $\frac{1}{r^b}$ for $b = 0.7878$. Examining the figures in [19] one can see that this fits well only for passwords of rank between 2^6 and 2^{20}. Bonneau [5] also explored the relationships between password distributions and zipf distributions.

Bonneau [6] criticized the comparability and repeatability of past password cracking results using tools such as John the ripper and/or dictionaries. He proposed metrics for studying the overall level of security in large password datasets, based only on the distribution, and not on the actual password strings. This work is thus largely orthogonal to ours. The metrics in [6] cannot be effectively applied to evaluating the security of a single, or a small dataset collected under a given scenario. We believe that our approach of using standard Markov models also corrects many of limitations identified in [6].

VII. Conclusions

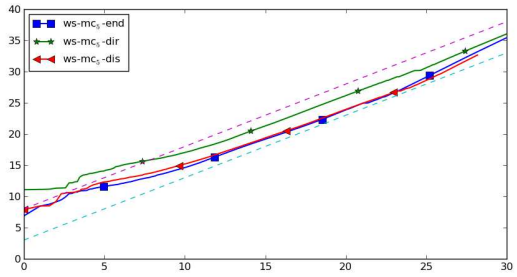
We make three contributions in this paper. The first is to introduce probability-threshold graphs for evaluating password datasets. The second is to introduce knowledge and techniques from the rich literature of statistical language modeling into password modeling. We also identify new issues (such as normalization) that arise from modeling passwords, and a broad design space for password models, including both whole-string models and template-based models. Third, we have conducted a systematic study of many password models, and obtained a number of findings. In particular, we show that the PCFG$_W$ model, which has been assumed to be the state of the art and has been widely used in password research, underperforms whole-string Markov models in our experiments. We expect that the new methodology and knowledge of effectiveness of Markov models can benefit future password research.

Acknowledgement. This paper is based upon work supported by the United States National Science Foundation under Grants No. 1314688 and 0963715, and by by United States Army Research Office Award 2008-0845-04 through North Carolina State University.

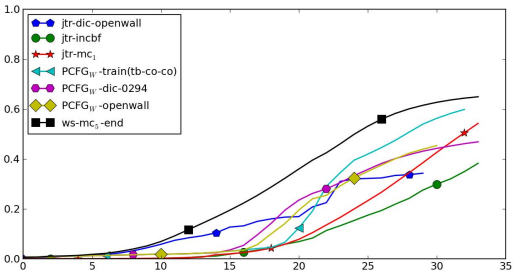
We would also like to thank Lujo Bauer, shepherd for this paper, and other reviewers for their helpful comments, which guided us revise and improve the paper.

References

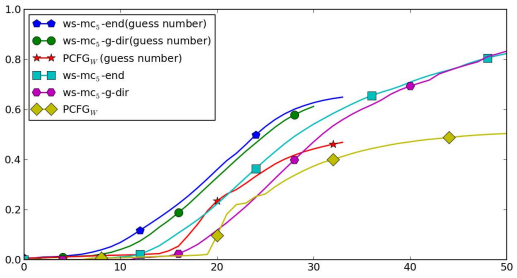
[1] Passwords, 2009. http://wiki.skullsecurity.org/Passwords.
[2] Csdn cleartext passwords, 2011. http://dazzlepod.com/csdn/.
[3] John the ripper password cracker, 2013. http://www.openwall.com/john/.
[4] Openwall wordlists collection, 2013. http://www.openwall.com/wordlists/.
[5] J. Bonneau. *Guessing human-chosen secrets*. Thesis, University of Cambridge, 2012.
[6] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 538–552. IEEE, 2012.
[7] W. E. Burr, D. F. Dodson, and W. T. Polk. *Electronic authentication guideline*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2004.
[8] C. Castelluccia, M. Dürmuth, and D. Perito. Adaptive password-strength meters from Markov models. In *Proceedings of NDSS*, 2012.
[9] M. Dell'Amico, P. Michiardi, and Y. Roudier. Password strength: an empirical analysis. In *Proceedings of INFOCOM*, pages 1–9, 2010.
[10] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, and C. Herley. Does my password go up to eleven?: The impact of password meters on password selection. In *Proceedings of CHI*, pages 2379–2388, 2013.
[11] D. Florencio and C. Herley. A large-scale study of web password habits. In *Proceedings of WWW*, pages 657–666, 2007.
[12] A. Forget, S. Chiasson, P. C. van Oorschot, and R. Biddle. Improving text passwords through persuasion. In *Proceedings of SOUPS*, pages 1–12, 2008.
[13] W. A. Gale and G. Sampson. Good-turing frequency estimation without tears. *Journal of Quantitative Linguistics*, 2(1):217–237, 1995.
[14] C. Herley and P. C. van Oorschot. A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy*, 10(1):28–36, 2012.
[15] S. Houshmand and S. Aggarwal. Building better passwords using probabilistic techniques. In *Proceedings of ACSAC*, pages 109–118, 2012.
[16] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400401, 1987.
[17] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *IEEE Symposium on Security and Privacy*, pages 523–537, 2012.
[18] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman. Of passwords and people: measuring the effect of password-composition policies. In *CHI*, pages 2595–2604, 2011.
[19] D. Malone and K. Maher. Investigating the distribution of password choices. In *Proceedings of WWW*, pages 301–310, 2012.
[20] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur. Measuring password guessability for an entire university. In *Proceedings of ACM CCS*, pages 173–186, Berlin, Germany, 2013. ACM.
[21] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of ACM CCS*, pages 364–372, 2005.
[22] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, pages 2:1–2:20, New York, NY, USA, 2010. ACM.
[23] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, et al. How does your password measure up? the effect of strength meters on password creation. In *Proceedings of USENIX Security Symposium*, 2012.
[24] M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 162–175, 2010.
[25] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *IEEE Symposium on Security and Privacy*, pages 391–405, 2009.
[26] Y. Zhang, F. Monrose, and M. K. Reiter. The security of modern password expiration: An algorithmic framework and empirical analysis. In *Proceedings of ACM CCS*, pages 176–186, 2010.



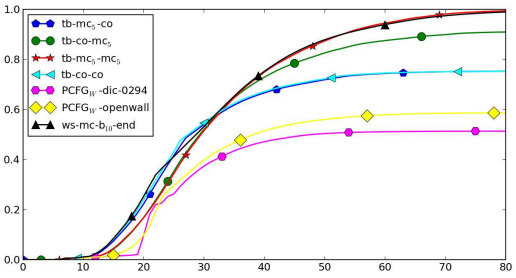
(a) Rank vs. probability: (x, y) denotes the 2^x most likely password has probability $\frac{1}{2^y}$; dashed lines are $y = x + 3$ and $y = x + 8$



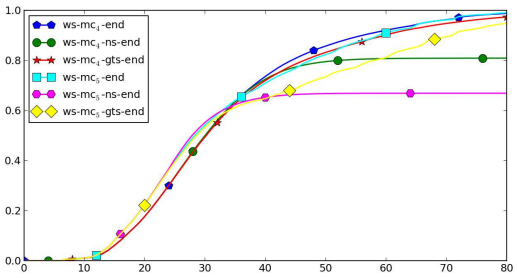
(b) Guess-number graph: (x, y) denotes 2^x guesses cover y portion of the dataset



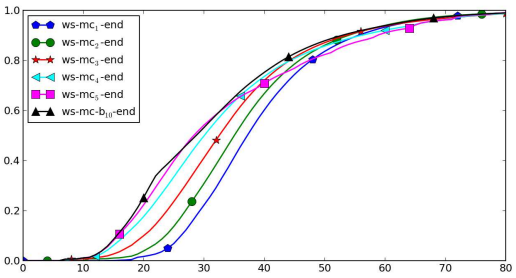
(c) Superimposing guess number and prob. threshold: for guess-number curves, x stands for 2^x guesses; for threshold curves, x stands for probability threshold $\frac{1}{2^x}$; dic-0294 is used as dictionary for PCFG$_W$



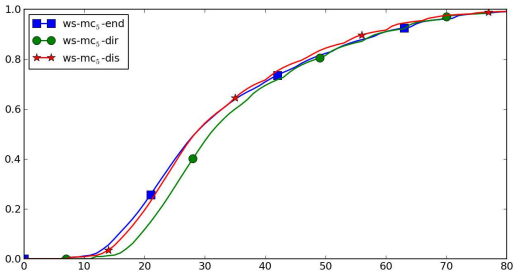
(d) Prob. threshold graph for comparing template-based models (including PCFG$_W$)



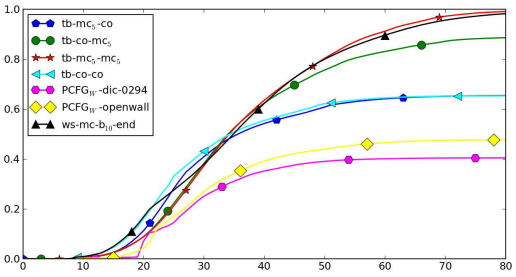
(e) Prob. threshold graph for comparing the effect of smoothing, all with end-based normalization



(f) Prob. threshold graph for comparing Markov of different orders; with end-based normalization and add-δ smoothing

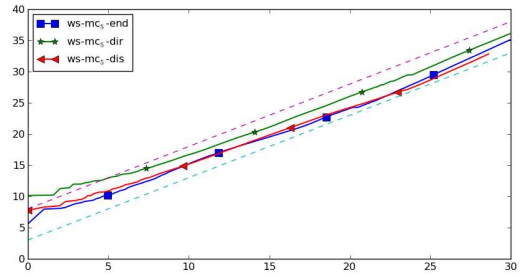


(g) Prob. threshold graph for comparing the effect of normalization

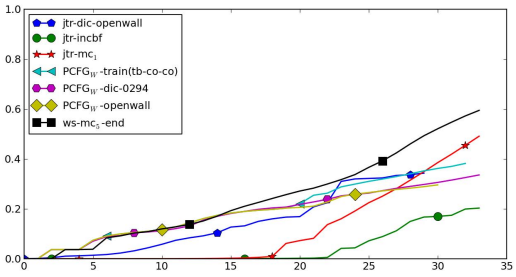


(h) Prob. threshold graph for passwords with length no less than 8; comparing template-based models (including PCFG$_W$)

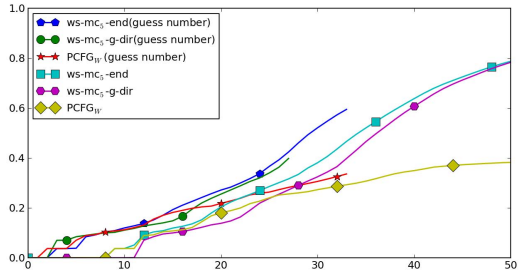
Fig. 2: Experiment result of Scenario 1: Rock→Ya+Ph.



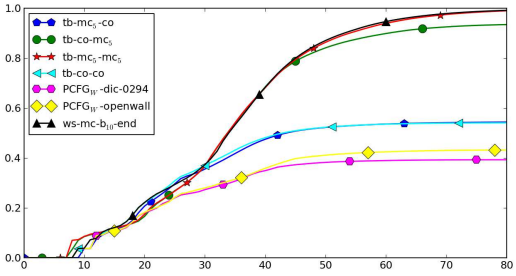
(a) Rank vs. probability: (x, y) denotes the 2^x most likely password has probability $\frac{1}{2^y}$; dashed lines are $y = x + 3$ and $y = x + 8$



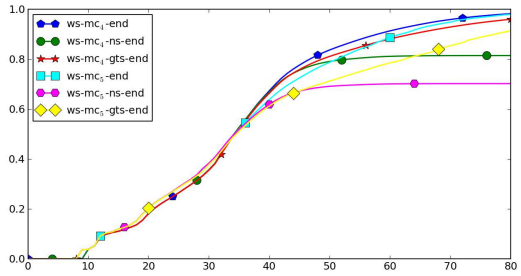
(b) Guess-number graph: (x, y) denotes 2^x guesses cover y portion of the dataset



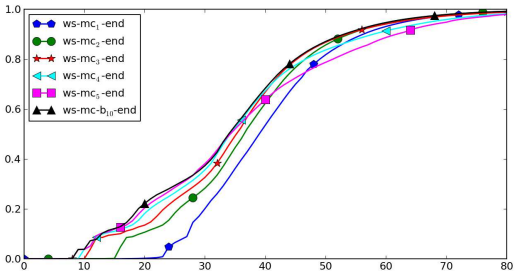
(c) Superimposing guess number and prob. threshold: for cracking curve, x stands for 2^x guesses; for threshold curve, x stands for probability threshold $\frac{1}{2^x}$; dic-0294 is used as dictionary for PCFG$_W$



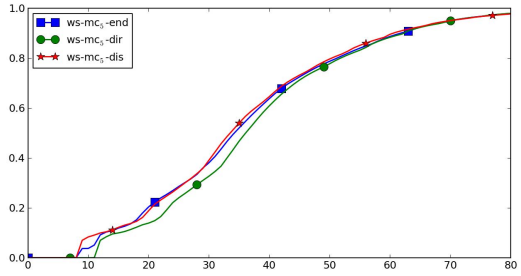
(d) Prob. threshold graph for comparing template-based models (including PCFG$_W$)



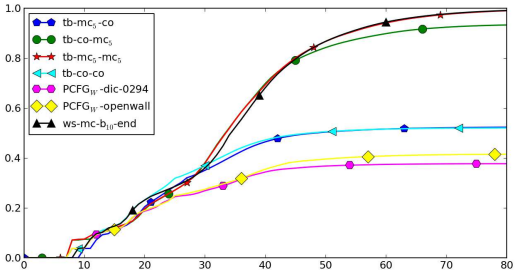
(e) Prob. threshold graph for comparing the effect of smoothing, all with end-based normalization



(f) Prob. threshold graph for comparing Markov of different orders; with end-based normalization and add-δ smoothing

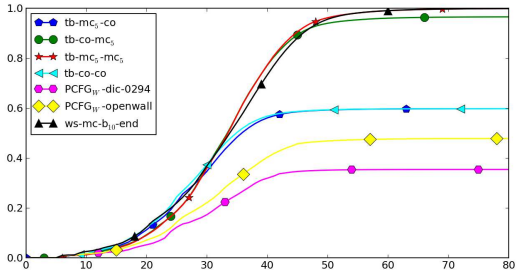


(g) Prob. threshold graph for comparing the effect of normalization

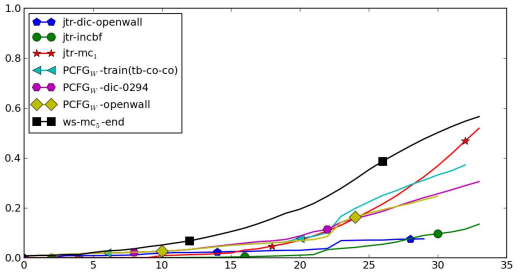


(h) Prob. threshold graph for passwords with length no less than 8; comparing template-based models (including PCFG$_W$)

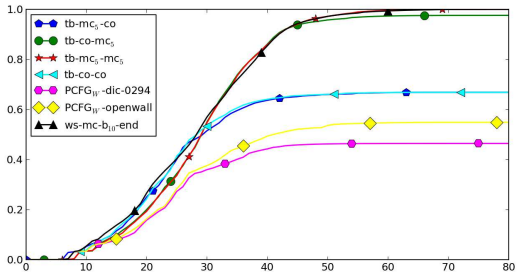
Fig. 3: Experiment results for Scenario 2: Du+178→CSDN.



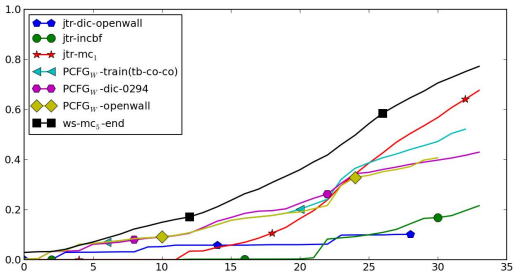
(a) Scenario 3: CS+178→Dudu: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



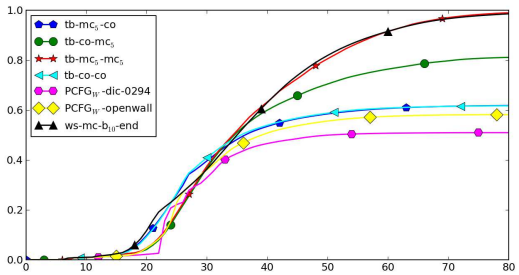
(b) Scenario 3: CS+178→Dudu: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset



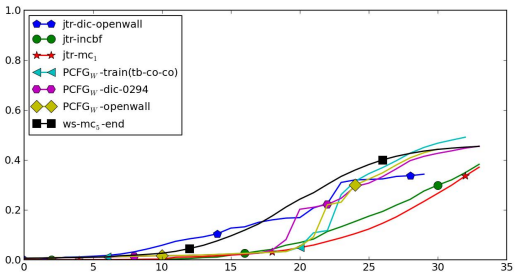
(c) Scenario 4: CS+Du→178: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



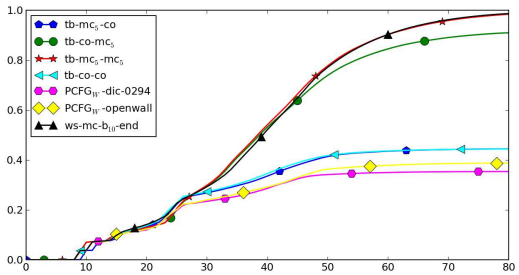
(d) Scenario 4: CS+Du→178: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset



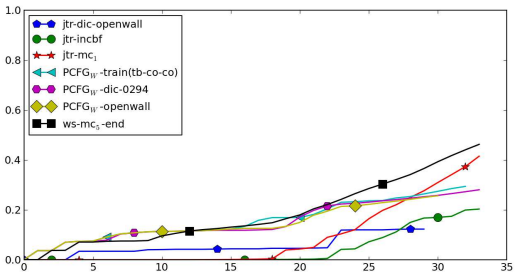
(e) Scenario 5: Chin→Ya+Ph: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



(f) Scenario 5: Chin→Ya+Ph: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset



(g) Scenario 6: Rock→CSDN: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



(h) Scenario 6: Rock→CSDN: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset

Fig. 4: Experiment result for Scenarios 3, 4, 5, 6.