Monte Carlo Strength Evaluation: Fast and Reliable Password Checking

Matteo Dell'Amico
Symantec Research Labs, France
matteo_dellamico@symantec.com

Maurizio Filippone[*]
University of Glasgow, UK
maurizio.filippone@eurecom.fr

ABSTRACT

Modern password guessing attacks adopt sophisticated probabilistic techniques that allow for orders of magnitude less guesses to succeed compared to brute force. Unfortunately, best practices and password strength evaluators failed to keep up: they are generally based on heuristic rules designed to defend against obsolete brute force attacks.

Many passwords can only be guessed with significant effort, and motivated attackers may be willing to invest resources to obtain valuable passwords. However, it is eminently impractical for the defender to simulate expensive attacks against each user to accurately characterize their password strength. This paper proposes a novel method to estimate the number of guesses needed to find a password using modern attacks. The proposed method requires little resources, applies to a wide set of probabilistic models, and is characterised by highly desirable convergence properties.

The experiments demonstrate the scalability and generality of the proposal. In particular, the experimental analysis reports evaluations on a wide range of password strengths, and of state-of-the-art attacks on very large datasets, including attacks that would have been prohibitively expensive to handle with existing simulation-based approaches.

1. INTRODUCTION

After years spent on unsuccessful attempts to overcome password-based authentication, researchers have come to accept that, in the foreseeable future, passwords will still be used in a large class of scenarios [12]. The recent public leaks of very large password datasets (the latest one in February 2015 involved ten million passwords *along with usernames* [3]) can only exacerbate the well-known security problems of password authentication.

Often, attackers can perform *guessing attacks*, i.e., attempt to guess a password by trying a large number of candidates. Examples include cases where an attacker wants to find the password that protects a wireless network or the master password that unlocks a password manager, or cases where the attacker has access to a list of hashed passwords or to a laptop where data is password-encrypted. Advanced probabilistic guessing schemes [18, 19, 21, 28, 30] use leaked password lists as training sets, improving their capability to guess even passwords that have not been leaked.

While passwords remain a key part of security infrastructures and attacks become more and more efficient, solutions designed to help users choose better passwords are still unsatisfactory. "Best practices," such as mixing uppercase and lowercase letters with digits, were conceived to defend users from brute-force attacks that have progressively become obsolete. Against current attacks, these practices strike bad trade-offs between usability and security [10, 11, 29].

A promising direction that motivates this work is represented by password meters. Evidence shows that users are influenced in their password choice when informed about their "strength" [9, 27]. Although password meters encourage users to choose better passwords, their output is often questionable as it is not a reliable assessment of the effort that attackers need to break a password [6].

These considerations suggest what the objective of password meters should actually be, and yield the following widely accepted definition of password strength [1, 7, 15, 18, 19, 28, 29]: password strength is defined as the *number of attempts that an attacker would need in order to guess it.* The definition underlies a guessing strategy, and indicates that it is possible to compute password strength by emulating it; however, such an approach is very expensive and – even after considerable computational efforts – the strength of a substantial fraction of unguessed passwords remains unknown. For this reason, as we discuss in Section 2, existing literature does not provide a satisfactory solution to the problem of efficiently evaluating password strength.

The best known guessing attacks adopt *probabilistic* approaches [18, 19, 21, 28, 30], which model ways users choose passwords, resulting in an assignment of a probability to any password. These probabilities are then used by guessing attacks to determine in what sequence passwords should be tried. Based on the above definition of password strength, this paper proposes a novel way to efficiently and accurately evaluate the strength of any password against a given probabilistic attack. Our method, described in Section 3, estimates password strength by *sampling* from the model, i.e., generating random passwords according to the probabilities assigned by the model. The appealing properties of our proposal are *computational efficiency* and *accuracy*. Computa-

[*]M. Filippone is now with EURECOM, France.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
© 2015 ACM. ISBN 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/XXX.XXXXXXX.

tional efficiency stems from the fact that the sampling step is cheap and can be precomputed; once this is done, computing password strength is as cheap as an array lookup through binary search. Accuracy can be rigorously characterized using the theory of Monte Carlo sampling, and a key result of the paper is that our estimate of password strength converges to the correct value with a convergence rate of $O\left(1/\sqrt{n}\right)$, n being the number of passwords in the sample.

We evaluate our method by considering very large datasets of leaked passwords, and state-of-the-art attacks such as the n-grams proposed by Narayanan and Shmatikov [21], the probabilistic context free grammars by Weir et al. [30], and the "backoff" model proposed by Ma et al. [19] (for which we propose an improvement). Details on the evaluation are available in Section 4.

In Section 5, we use our method to perform an extensive empirical evaluation. In the first part of Section 5, we empirically analyze the precision of our method: we find that sample sizes as small as 100-1,000 are sufficient to obtain password strength estimates in the right order of magnitude, which is suitable for the implementation of a strength meter. More accurate results can simply be obtained by increasing the sample size, yielding a relative error around 1% when the sample size is 100,000.

In the second part of Section 5, we use our method to compare the performance of attack models, overcoming the limitations of existing studies, as discussed in Section 2.2. We study passwords that are extremely difficult to guess (requiring up to 2^{80} attempts), and we find that no approach is a clear winner. Depending on the number of guesses that an attacker can afford, different approaches become preferable.

In the third part of Section 5, we analyze the importance of training sets with respect to attack effectiveness; we show that larger training sets improve attacks against "average" passwords, whereas not much is gained for passwords that are either particularly easy or hard to guess.

We conclude Section 5 by assesssing the impact of traditional restrictions (i.e., limitations on length or classes of character composing the password) by evaluating the improvements in password strength that such restrictions can obtain: the results suggest that length requirements are more advisable than those about character composition.

Contributions. Our main contributions are:

1. A sound method to compute password strength, according to the consensus definition of robustness against guessing attacks. The method is lightweight and easy to implement, and we provide theorems to prove its correctness and approximation level.

2. An empirical evaluation of the accuracy of our method, including the trade-off between computation cost and precision. We show that accurate estimations can be obtained at a low computational cost.

3. An extensive empirical evaluation comparing state-of-the-art attack models, impact of training set and of restrictions in password choice. Our method allows performing this analysis while overcoming the limitations of previous research discussed in Section 2.2.

2. RELATED WORK

We first outline the state of the art in terms of password guessing in general (Section 2.1); we then focus on studies that gauge password strength, highlighting the limitations this work improves on (Section 2.2).

2.1 Password Guessing

The first studies on password guessing attacks date back to 1979, when Morris and Thompson [20] reported that it was already possible for computers to guess "a substantial fraction" of the passwords that were used in Unix systems through brute force and dictionary attacks; similar studies after more than two decades show that not much has changed in the meanwhile [16, 26].

Rainbow chains [22] are a technique that allows to efficiently memorize a very large set of pre-computed password hashes and find passwords that appear in them. They are defeated by the technique of *salting*, i.e. appending a random string of bits to passwords before computing their hash.

More recently, probabilistic attacks have been proposed to drastically reduce the number of guesses for passwords that are long and/or do not appear in dictionaries: notable examples are attacks based on n-gram models [21] and probabilistic context-free grammars (PCFGs) [30]. These approaches build a model through a training set of passwords in cleartext; password creation is then seen as a stochastic process where each password has a given probability of being chosen. To minimize the number of needed guesses, probabilistic attacks enumerate the guesses by descending probability. Recent improvements to these attacks include the proposal of a backoff technique to improve n-gram models [19] and amending PCFGs to include semantic patterns [28] and to better suit Chinese passwords [18]. In this work, we implemented the n-grams, PCFGs and backoff models; they are described in detail in Section 4. In Section 5.2, we provide extensive experimental results to compare them.

A key defense technique against guessing attacks is password *strengthening*, or *stretching*, which amounts to hashing passwords using computationally expensive functions, resulting in a slowing down of guessing attacks. The design of strengthening techniques that are resilient to attacks that use parallelization is an active topic of research [23, 25]. Strengthening is a tool that essentially multiplies the strength of a password by a constant factor, and this benefit is counterbalanced by the inconvenience of additional computation whenever a legitimate user's password is checked: better knowledge of password strength allows to better choose a desirable point in this trade-off.

2.2 Password Strength Evaluation

The ubiquity of password authentication makes it obviously important to evaluate password strength, i.e., how difficult it is to guess them. Traditionally, simple strategies based on the number of characters and the presence of special characters such as uppercase characters, digits and symbols have been used [4]; however, these approaches have been found to be inadequate to quantify the resistence against modern attacks [29]. Indeed, recent studies evaluate password strength as the number of attempts an attack would need to guess it. With this metric, a password of strength 2^x can be considered as strong as a symmetric encryption key of x bits, since an attacker can guess either with the same effort.

Dell'Amico et al. [7] adopted an approximation technique to evaluate the number of attempts needed to guess passwords when using n-gram models. Besides being limited to n-grams, this technique has scalability issues with the large state space produced by state-of-the art attacks with $n \geq 3$ and datasets having millions of passwords. Our proposal requires a fraction of the resources and is applicable in general to any probabilistic password model.

Bonneau [1] studied a large password dataset to consider the *ideal* case of probabilistic attacks that perfectly capture the probability with which users choose passwords. This approach allows characterizing the strength of the passwords that appear multiple times in the dataset, but the complete probability distribution remains unknown: Bonneau reports that Kolmogorov-Smirnov tests rejected the interpolation to a power-law distribution with high confidence. Unfortunately, these considerations imply that the behavior of ideal probabilistic models is still uncertain for the less frequent (and hence stronger) passwords that most users choose.

Kelley et al. [15] measured the strength of passwords against Brute Force Markov (BFM) and PCFGs. BFM is a hybrid between brute force cracking and n-gram models for which computing exactly the number of guesses needed is easy; unfortunately, BFM performs definitely worse than n-gram models that are actually used by attackers, finding very few passwords within the first $2^{40} - 2^{45}$ attempts. For PCFGs, Kelley et al. employed a 64-node Hadoop cluster for several days to emulate an attack of around 2^{45} guesses; our approximated approach, instead, returns an accurate account of password strength in a fraction of a second even for passwords that require around 2^{80} attempts to be found.

Ma et al. [19] studied probabilistic password models such as n-grams and PCFGs, and proposed a new "backoff" model that we consider in the evaluation section of this paper. Rather than attempting to simulate very expensive attacks, Ma et al. resorted to analyzing the distribution of probabilities that existing models associate to large password datasets. As the authors themselves acknowledge, this technique is useful in "type-1" research where passwords in different datasets are compared against the same attack model, but it can give misleading results in "type-2" research where different attack models are compared against the same password dataset, because two passwords that are assigned the same probability by two different attack models may be guessed after a different number of attempts in the two attacks: for example, a password with probability 2^{-15} could be the very first guess in a model and the thousandth one in a different one. Ma et al. still performed "type-2" comparisons, based on the conjecture of a roughly linear relationship between probability and password strength. In Section 5.1, we show that this conjecture does not hold true for a variety of datasets and attack models; this motivates the importance of our proposal that aims at computing password strength in terms of number of guesses rather than probabilities.

Password strength is typically presented to users through "strength meters". Studies show that users put in practice the suggestions given by strength meters in order to generate stronger passwords [9,27]; unfortunately, strength meters are generally based on heuristic methods that do not necessarily reflect the resistance of passwords to guessing attacks [6]. Castelluccia et al. [5] show the design of a strength meter that outputs strength as the probability that a Markov model would assign to a password; as argued above, probability does not give a clear information about the number of guesses needed to break a password: our mechanism can be integrated in this framework to transform the password checker's output from a probability to the number of guesses.

Telepathwords [17] is a system that highlights easy-to-guess patterns as users type their passwords: we think this approach is ideally complementary to a good password meter, helping users understand the reason *why* a password is assigned a given strength.

3. EVALUATING PASSWORD STRENGTH

State-of-the-art password research is based on probabilistic password models. Such models attempt to characterize the way humans choose their passwords by constructing a mapping between strings and the frequency with which humans are assumed to choose them as passwords. Let us denote the (possibly infinite) set of all allowed passwords as Γ; a probabilistic password model is a function p such that

$$\sum_{\alpha \in \Gamma} p(\alpha) = 1.$$

We call $p(\alpha)$ the *probability* of password α under model p.

Attack models enumerate passwords by descending order of probability: hence, the *strength* (or *rank*) $S_p(\alpha)$ of a password α under model p is the number of passwords that have probability higher than α:

$$S_p(\alpha) = |\{\beta \in \Gamma : p(\beta) > p(\alpha)\}| . \qquad (1)$$

We are interested in approximating $S_p(\alpha)$ efficiently and accurately. In addition to being able to compute $p(\alpha)$, our only requirement is to be able to generate a sample (with replacement) of passwords such that, at any draw, the probability of choosing password α is exactly $p(\alpha)$. Implementing this is not difficult for any of the models that we consider in this work (n-grams, PCFGs and backoff).

In order to use p for a guessing attack, the attacker needs to enumerate passwords by descending probability; doing this efficiently is an open problem [8, 21]. Our method, instead, relies only on sampling and does not require implementing this enumeration.

3.1 The Method

Evaluating password strength as defined in Equation 1 entails computing the cardinality of the set $\Delta \subset \Gamma$ of all passwords weaker than α, i.e., $\Delta = \{\beta \in \Gamma : p(\beta) > p(\alpha)\}$. Computing $S_p(\alpha)$ exactly would require generating all elements of Δ, which is obviously impractical if Δ is large. Our method approximates $S_p(\alpha)$ efficiently based on a subset of n passwords; we prove its convergence to the true value of $S_p(\alpha)$ for n going to infinity. In Section 3.2, we show how this method can be implemented efficiently, yielding a precomputed probability vs. rank curve.

We generate a sample Θ of n passwords. Passwords are sampled with replacement and each password β is sampled with probability $p(\beta)$. Then, our estimation C_Δ for the cardinality of Δ is

$$C_\Delta = \sum_{\beta \in \Theta} \begin{cases} \frac{1}{p(\beta) \cdot n} & \text{if } p(\beta) > p(\alpha), \\ 0 & \text{otherwise.} \end{cases} \qquad (2)$$

Note that the values of $p(\beta)$ can be very low, and these calculations may overflow or underflow. In our experiments, we observed these phenomena with 32-bit floating point values, and we avoided them by using 64-bits arithmetic. To avoid such problems altogether, one may want to resort to arbitrary-precision numeric implementations.

In the following, we prove two theorems: first, the expected value of C_Δ under the attack model p is $|\Delta|$, meaning that the expected value of our estimation is indeed the desired quantity in Equation 1; second, the variance decreases as the sample size grows, with a convergence rate of $O\left(1/\sqrt{n}\right)$ – the standard for Monte Carlo estimators.

Theorem 1. *The expected value of C_Δ is $|\Delta| = S_p(\alpha)$.*

Proof. Let us first consider the $n = 1$ case. Here,

$$E\left[C_\Delta\right] = \sum_{\beta\in\Gamma} p\left(\beta\right) \begin{cases} \frac{1}{p(\beta)} & \text{if } \beta \in \Delta \\ 0 & \text{otherwise} \end{cases} = \sum_{\beta\in\Delta} 1 = \left|\Delta\right|.$$

For $n > 1$, it is sufficient to note that our estimation is the mean of n estimates done with $n = 1$. Since all estimates are i.i.d., the expected value of the average is again $|\Delta|$. □

Theorem 2. *The standard deviation of C_Δ is $O\left(1/\sqrt{n}\right)$.*

Proof. The variance of a random variable X is $E\left[X^2\right] - \left(E\left[X\right]\right)^2$. For the $n = 1$ case, we compute the variance as

$$\begin{aligned}\mathrm{Var}\left(C_\Delta\right) = E\left[C_\Delta^2\right] - E\left[C_\Delta\right]^2 &= \sum_{\beta\in\Delta} \frac{p\left(\beta\right)}{p\left(\beta\right)^2} - \left|\Delta\right|^2 \\ &= \sum_{\beta\in\Delta} \frac{1}{p\left(\beta\right)} - \left|\Delta\right|^2.\end{aligned}$$

We notice that the above value is finite because, by hypothesis, $|\Delta|$ is always finite; let us call this value V_1. Since the variance of an average of n i.i.d. variables is the variance of those variables divided by n [2, Section 2.11.1], the generic expression for the variance will be

$$\mathrm{Var}\left(C_\Delta\right) = \frac{V_1}{n} = \frac{\sum_{\beta\in\Delta}\frac{1}{p(\beta)} - \left|\Delta\right|^2}{n}. \quad (3)$$

Since V_1 does not depend on n we obtain that, when n grows, the variance converges in probability to 0 and C_Δ converges in probability to $|\Delta|$. With a variance in $O\left(1/n\right)$, the standard deviation is $O\left(1/\sqrt{n}\right)$, which is what typically happens with Monte Carlo estimators [24]. □

In Equation 2, we are performing an inverse probability weighting: we multiply the contribution of each element in the sample by the inverse of the probability that such element has to appear in the sample. This procedure is not new in Statistics: it has been proposed in 1952 by Horvitz and Thompson [13] for stratified sampling – a technique where the analyst has control over sampling, and chooses different sampling probabilities for different subpopulations (strata). In our case, the difference lies in the fact that we do not have control on how to sample from Γ, and we assume the sampling probabilities as given. Moreover, since Γ is extremely large and possibly infinite, even assuming that uniform sampling was possible, elements from Δ in the sample would be very rare or non existent; on the other hand, since Δ includes elements with higher probabilities, they are definitely more likely to appear in our non-uniform sample.

3.2 Optimized Implementation

The method described so far would require to create a sample Θ of size n from the model p every time we want to evaluate the strength of a password α. This is not necessary: in the following, we show how to precompute the probability-rank curve, and compute the strength of a new password simply by performing an $O(\log n)$ lookup.

The key idea here is that the sampling step can be performed just once, and it is only necessary to store the probabilities $p\left(\beta_i\right)$ for each password β_i in Θ. We store the probabilities in an array $A = \left[p\left(\beta_1\right), \ldots, p\left(\beta_n\right)\right]$ sorted by descending probability, and we create an array C such that C's i-th element is

$$C\left[i\right] = \frac{1}{n}\sum_{j=1}^{i}\frac{1}{A[j]} = C\left[i-1\right] + \frac{1}{n \cdot A[i]}.$$

A probability $A[i]$ indeed corresponds to a rank $C[i]$. To compute the strength of a password α, we first lookup the index j of the rightmost $A[j]$ value in A such that $A[j] > p\left(\alpha\right)$ through binary search; the output for C_Δ is simply $C[j]$. The procedure has $O\left(\log n\right)$ cost due to binary search.

In Section 5.4, to evaluate mandatory restrictions on passwords, we evaluate the cardinality of a different set of passwords, a Δ' set of passwords that is defined via a boolean filter function f that identifies allowed passwords, such that $\Delta_f = \{\beta \in \Gamma : p\left(\beta\right) > p\left(\alpha\right) \wedge f\left(\beta\right)\}$. In this case, it is pointless to store information about passwords for which $f\left(\beta\right)$ is false, as they will never be taken in consideration when evaluating C_{Δ_f}; therefore, our array A will only contain the probabilities under model p of passwords that satisfy $f\left(\beta\right)$.

4. EXPERIMENTAL SETUP

Here, we describe the setup we employ to evaluate our proposal. We describe the datasets used in our evaluation and the state-of-the-art guessing techniques that we consider.

4.1 Datasets

Several very large password datasets have been made publicly available through leaks: the *de facto* standard in password research – allowing to compare results between different works – is the Rockyou dataset, which contains a set of 32 million passwords that was released to the public in 2009. On this dataset we measure password strength as follows. Whenever a password training set is needed, we use a dataset of 10 million passwords that was recently released (February 2015) on the Xato.net website [3] – in the following, we will label this as the Xato dataset. Most passwords in the Xato dataset appear to come from Western users, with the most common language being English. The Xato dataset does not include passwords from the Rockyou leak.

We performed additional experiments – not reported here due to space limitations – where we switched the role of datasets, using Xato as test set and Rockyou as a training set. Results are largely analogous to those reported here, reinforcing our confidence on the generality of our results, confirming that there are no discernible defects in either dataset, and showing that very large password datasets from users speaking the same languages exhibit similar properties.

We have chosen Xato and Rockyou because of dataset quality, uniformity and size. Other large datasets of Web passwords include the Chinese datasets studied by Li et al. [18]: the authors commented that “the raw files contain duplication and blank passwords that can affect the analysis". To avoid the risk of misleading results due to improper data cleaning, we left those datasets out of our analysis.

Other large datasets of more than one million passwords have been leaked from Gawker, eHarmony, LinkedIn, Evernote and Adobe. These datasets, however, contain hashed or encrypted passwords: therefore, they are not suitable for our analysis which needs passwords in clear-text to train the attack models and to evaluate strength.

4.2 Attack Models

We now describe the attack models we consider in the effort of covering the most representative approaches.

4.2.1 N-Grams

Password guessing attacks using n-grams (i.e., substrings of length n appearing in a training set) have been originally proposed by Narayanan and Shmatikov [21].

In the following, we denote the number of occurrences of a string of characters $c_a \dots c_b$ in the training set as $o(c_a \dots c_b)$. We further denote the frequency with which character c_b follows the string $c_a \dots c_{b-1}$ as

$$P\left(c_b | c_a \dots c_{b-1}\right) = \frac{o\left(c_a \dots c_b\right)}{o\left(c_a \dots c_{b-1}\right)} \qquad (4)$$

In an n-gram model (equivalently known as *order $n-1$ Markov model*), the probability of a password $c_1 \dots c_l$ is

$$p_{n\text{-gram}}\left(c_1 \dots c_l\right) = \prod_{i=1}^{l+1} P\left(c_i | c_{i-n+1} \dots c_{i-1}\right), \qquad (5)$$

where all values c_i when $i \leq 0$ or $i > l$ are considered to be a special symbol $\perp$ that does not appear in passwords, which is used to denote the start or the end of the password.

Higher values of n make it possible to exploit a longer history when predicting the next character; however, as n grows, the issue of *data sparsity* appears: some n-grams may not be represented in the training set, and the model would incorrectly label their probabilities as 0. A solution to this problem is the backoff model described in Section 4.2.3.

Implementation Details.

For each n-gram $c_{i-n+1} \dots c_i$, we simplify the notation by calling $c_{i-n+1} \dots c_{i-1}$ as its *state* s and c_i as its *transition* t. We precompute all values of $P\left(t|s\right)$ by grouping all the n-grams by state: in this way, the number of occurrences of the state s needed for Equation 4 is computed by summing the occurrences of all the n-grams in the group.

The probabilities that we compute can be very small and may underflow: to avoid such problems, we store and compute the base-2 logarithms of probabilities rather than probabilities themselves. When computing probabilities, we perform similar substitutions also for PCFG and backoff methods described later in the text.

We also precompute data to facilitate sample creation and password enumeration: for each state s, we sort all transitions $t_{s,i}$ in descending order of number of occurrences; we then precompute an array C_s of cumulative probabilities such that each element $C_s[i] = \sum_{j=1}^{i} P\left(t_{s,j}|s\right) = C[i-1] + P\left(t_{s,i}|s\right)$. The creation of a password for the sample needed in Section 3.1 is then carried out according to the procedure shown in Algorithm 1: using binary search on C_s speeds up password generation.

Algorithm 1 Password generation for n-grams.

```
def starting_state():
    return "$\perp \dots \perp$" with length $n-1$
def update_state($s, t$):
    drop the first character from $s$
    return $s + t$ # concatenation
$s \leftarrow$starting_state()
$g \leftarrow$"" # accumulator for the result
while True:
    $r \leftarrow$random number in $[0, 1]$
    # find $i$ through binary search
    $i \leftarrow$rightmost index s.t. $C_s[i] > r$
    if $t_{s,i} = \perp$: return $g$
    append $t_{s,i}$ to $g$
$s \leftarrow$update_state($s, t_{s,i}$)
```

We also implemented a method to generate explicitly the passwords with highest probability, in order to evaluate empirically the accuracy of our approach as done in Section 5.1. To keep memory usage under control we use a depth-first exploration of the state space rather than using a priority queue, exploring all states that have probability higher than a chosen threshold [19]; we sort passwords by decreasing probability after having generated them.

4.2.2 PCFGs

Probabilistic Context-Free Grammars (PCFGs) have been proposed by Weir et al. in 2009 [30]: they are based on the realization that passwords often follow similar structures, such as for example a word followed by a number.

In the PCFG approach, passwords are grouped by *templates*, which are the way sequences of letters, digits and symbols are concatenated to form the password. For example, the "abc123" password has the L_3D_3 template, meaning that it is composed of three letters followed by three digits. The probability of the password is calculated by multiplying the frequency of the template $P\left(L_3D_3\right)$ by the frequency of each pattern in the template, i.e.,

$$p_{\text{PCFG}}\left(\text{“abc123”}\right) = P\left(L_3D_3\right) P\left(\text{“abc”}|L_3\right) P\left(\text{“123”}|D_3\right),$$

where $P\left(\text{“abc”}|L_3\right)$ and $P\left(\text{“123”}|D_3\right)$ are the frequency of "abc" and "123" within the set of three-characters groups of, respectively, letters and digits.

Weir et al. proposed to obtain the frequencies of templates, digits and symbols from a training set of passwords and to get the frequencies of letter groups from a dictionary; however, when the training set is large, PCFGs perform better by calculating also letter group frequencies from the training set [19]. In Section 5.2, we confirm this result.

Implementation Details.

We implemented PCFGs following the specifications by Weir et al.; implementing the sampling step is trivial.

Instead of storing the frequencies of templates and groups of characters, we store their number of occurrences. This allows us to perform evaluations where the test set and the training set are the same with "leave-one-out" cross-validation: when computing the probability for a given password, we subtract the one occurrence which is due to the password under evaluation.

4.2.3 *Backoff*

The backoff model has been proposed by Katz in the field of natural language processing [14], and it has been proposed for password cracking by Ma et al. [19]. This model addresses sparsity by considering n-grams of *all* lengths, and discarding those with less occurrences than a threshold τ. Intuitively the idea is that, for each choice of a new character, the model considers the longest sequence of past characters appearing at least τ times in the training set.

Given a threshold τ, the probability of a password $c_1 \dots c_l$ under the backoff model can be defined by induction, starting from the single character case. The probability of a single character $\hat{c}$ is that character's frequency in the training set:

$$p_{\text{bo}}\left(\hat{c}\right) = \frac{o\left(\hat{c}\right)}{\sum_c o\left(c\right)},$$

where – as in Section 4.2.1 – $o\left(c_1 \dots c_n\right)$ is the number of occurrences of the $c_1 \dots c_n$ string in the training set. The probability of a string of more than one character is

$$p_{bo}\left(c_1 \dots c_{n+1}\right) = p_{\text{bo}}\left(c_1 \dots c_n\right) \cdot P\left(c_{n+1} | c_1 \dots c_n\right),$$

where

$$P\left(c | c_1 \dots c_n\right) = \begin{cases} \frac{o(c_1 \dots c_n c)}{o(c_1 \dots c_n)} \text{ if } o\left(c_1 \dots c_n c\right) \geq \tau, \\ P\left(c | c_2 \dots c_n\right) r\left(c_1 \dots c_n\right) \text{ otherwise} \end{cases}$$

and

$$r\left(c_1 \dots c_n\right) = \sum_{c : o(c_1 \dots c_n c) \geq \tau} \frac{o\left(c_1 \dots c_n c\right)}{o\left(c_1 \dots c_n\right)}.$$

As for the n-grams case, each password is considered to end with a $\perp$ symbol.

We noticed that, by construction, this model often generates passwords that are suffixes of common ones (e.g., "ssword"). To avoid this problem, we propose to prepend – similarly to what happens for n-grams – a start symbol $c_o = \perp$ to each password. We will see in Section 5.2 that this modification improves the quality of the guesses for this model.

Implementation Details.

The output of this method only depends on the number of occurrences of n-grams with at least τ occurrences. A naïve way of implementing this model would process the training set once, count the number of occurrences of all n-grams for any value of n and, at the end of the process, drop information for all n-grams with less than τ occurrences. This approach requires very large amounts of memory, most of which is devoted to temporarily memorizing the (in most cases low) number of occurrences for n-grams with large n. It is therefore problematic for our very large datasets.

In our implementation, we proceed by scanning the training set multiple times, and at the i-th iteration we compute the number of occurrences of i-grams, discarding those that have less than τ occurrences. We know for sure that the number of occurrences of an i-gram will not be higher than the number of occurrences of both its prefix and suffix of length $i-1$: therefore, we can avoid storing any information about i-grams whose prefix or suffix is not included in the previous iteration. As an additional optimization, at the i-th iteration we avoid considering passwords that – thanks to this optimization – had no impact on the calculation in the previous round.

Algorithm 2 Sample creation for the backoff model.

```
def starting_state():
    if using the start symbol:  return "$\perp$"
    else: return ""
def update_state($s, t$):
    append t to s
    while $o(s) < \tau$:
        drop the first character in $s$
Run Algorithm 1 using these functions.
```

We store the number of occurrences of each n-gram to facilitate leave-one-out cross validation, for the reasons described in Section 4.2.2. However, to make it efficient to create a sample and to generate passwords, we also precompute the values of each $P\left(c | c_1 \dots c_n\right)$ when $o\left(c_1 \dots c_n\right) \geq \tau$. We consider $c_1 \dots c_n$ as a state and c as a transition, and we apply the same treatment described in Section 4.2.1 to generate the C_S auxiliary data structures. The sample creation algorithm then follows the procedure introduced for n-grams (Algorithm 1), overriding the `starting_state` and `update_state` functions as described in Algorithm 2.

5. EXPERIMENTAL RESULTS

We now proceed with our experimental evaluation, in which we provide the following contributions: we evaluate the precision – and the limitations – of our method (Section 5.1). We then provide a detailed comparison of the efficacy in cracking passwords of the attack models we consider (Section 5.2); we discuss the impact of training set size on the performance of guessing algorithms (Section 5.3); finally, we evaluate the strength of passwords that satisfy typical mandatory restrictions (Section 5.4).

Unless otherwise stated, in the experiments the sample size $n = |\Theta|$ used by our method is 10,000; results are obtained by evaluating a random subset of 10,000 passwords in the test (Rockyou) database; the value of τ for the backoff model is 10 in line with Ma et al. [19]. Models are always trained on the *full* training datasets.

5.1 Method Precision

In Figure 1, we show the relationship between the probability that models attribute to passwords and their rank (i.e., the $S_p\left(\alpha\right)$ value introduced in Equation 1). We evaluate the models that behave best for password guessing (more information about them is provided in Section 5.2). For each of these models, we generated all the passwords having probability higher than 2^{-20}; for each of them, we plot their rank against their probability. The shaded areas are the ones between the minimum and maximum estimation provided by our model, re-generating a sample Θ for 5 different runs.

In the first plot, estimates are so close to the correct values that it is impossible to discern the shaded areas; in the second plot, we show a closeup of a particular area. We experimentally confirm the result we have proven in Theorem 1: our estimates converge to the correct values. An interesting insight that we gain by looking at the leftmost area of the password enumeration space, is that the probability-rank relations can clearly differ between models.

The difference in the probability vs. rank plot is due to the fact that each model generates passwords with a different probability distribution. In Figure 2, we show the

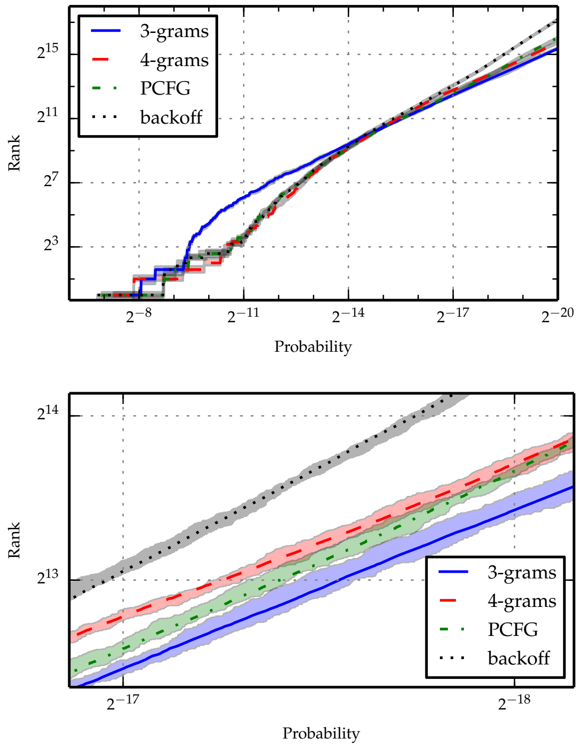


Figure 1: Password probability vs. rank. We show the correct values obtained by enumerating passwords by descending probabilities and, in the shaded areas, the area between the minimum and maximum of 5 different estimates.

histograms resulting from binning the generated passwords in each $\left[2^{-n}, 2^{-n-1}\right)$ probability interval. From these plots, it is clear that models have a very different signature in terms of probabilities attributed to passwords: in particular, the probability distribution of passwords generated through PCFGs and the backoff model are clustered on probabilities between 2^{-20} and 2^{-25}; n-gram models, instead, are more likely to generate passwords that have very low probabilities.

It is interesting to note that the training set contains around 2^{23} unique passwords; as we shall see in the following results, this means that passwords labeled with a probability lower than around $2^{-27} - 2^{-29}$ are unlikely to be present in the original datasets. Models such as 3- and 4-grams are definitely more likely to generate, when sampled, passwords that do not appear in the original training set.

Theorem 1 proves that our estimates are correct on average, but the expression of the variance in Equation 3 is difficult to interpret, because it depends on an unknown probability distribution. In Figure 3, we plot the level of uncertainty we have in our estimation, in the form of relative sample standard deviation.[1] We plot the results obtained on evaluations based on 30 different samples, using the 3-gram model. This plot helps us in quantifying the precision in estimating password strength and shows that, as predicted by Theorem 2, estimates do converge as the sample size grows. These results can be used as a guideline to choose the sample size n: if the user is only interested in a rough order-of-magnitude estimation of password strength, even a sample that is as small as 100 elements can be sufficient.

In Figure 4, we investigate the limitations of our approach. When we start evaluating very strong passwords, our estimation of their strength becomes definitely more uncertain: the reason is that, if $p(\alpha)$ is low, a password with probability close to $p(\alpha)$ is unlikely to appear in our sample, and its presence or absence has a strong impact on our strength estimation since it would add a (large) value $1/p(\alpha)$ to the estimation. The PCFG and backoff models are less likely to generate passwords with very low probability (see Figure 2), and this results in higher uncertainty on their strength. Strength estimation is less precise for stronger passwords: when these estimates are used to inform users, we deem this uncertainty acceptable, as these passwords can generally still be considered strong enough for most evaluation purposes.

In Figure 5, we compare the rank–probability pairs for ranges that are expensive or prohibitive to evaluate by generating explicitly the passwords (in this case, to limit the uncertainty of our estimation for very small probabilities, we raised the sample size to 100,000). The probability values diverge between models as the rank grows: the probabilities for passwords having the same rank can differ between models by orders of magnitude. This result suggests that it is dangerous to compare probability attributed to passwords by different models when what we are interested in is password strength; our method, instead, provides a sound solution to this problem.

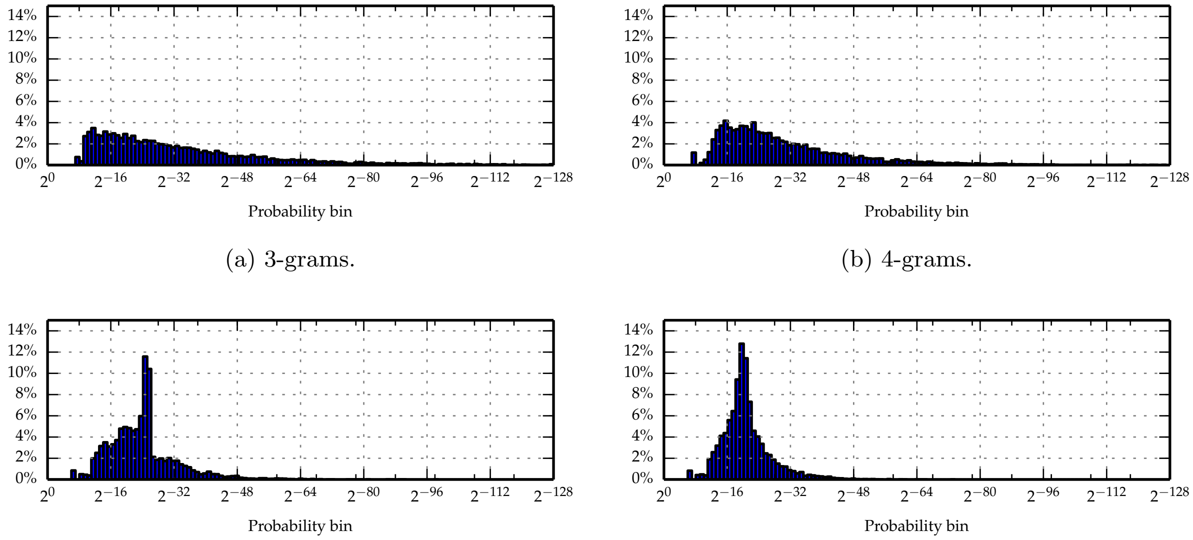
In Figure 6, we highlight the difference between the models by plotting the value of probability *multiplied by* rank. If models had similar rank values associated with the same probability, this value should be similar between attack models; when discussing the opportunity of comparing probabilities between different attack models, Ma et al. [19] conjectured that this value should remain between 2^{-3} and 2^{-8}; this would have implied a roughly linear relationship between probability and rank, and justified comparing probabilities output by different models when ranks are expensive to compute. Our results show that this is not the case: again, this comes as evidence that comparing the number of guesses is unavoidable if one wants to compare different probabilistic password models.

5.2 Comparing Attack Models

With the results of the previous section, we have established that our method yields precise estimates of password strength and that simply comparing probabilities – as done in other works – is not sufficient to reliably compare attacks. We now show a comparison in terms of *guess-number graphs*, plotting the probability that an attacker would guess a single password against the number of guesses; to the best of our knowledge, this study is the first to report guess-number graphs comparing several state-of-the-art attacks up to an extremely large (2^{80}) number of attempts. Here and in the following, results are obtained by evaluating the passwords in the Rockyou dataset, using Xato as a training set.

Figure 7 shows an overview of the attacks that – as we will see in the following – perform best. For the first $2^{16} - 2^{20}$ guesses, backoff and PCFG perform almost equivalently: this is because both models are essentially guessing the most frequent passwords in the training dataset. When the num-

[1] *Relative sample standard deviation* is the square root of sample variance (i.e., variance computed using Bessel's correction) divided by the mean of the values.



(a) 3-grams.

(b) 4-grams.

(c) PCFG.

(d) Backoff.

Figure 2: Probability distribution for passwords generated by different models. Each bin contains passwords having probabilities in the $\left[2^{-n}, 2^{-n-1}\right)$ interval. The probability distribution varies widely depending on the model used.

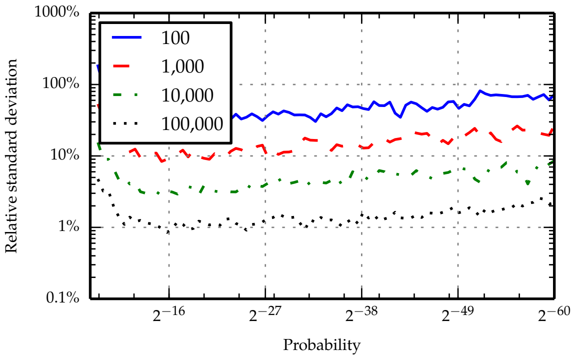


Figure 3: Estimation uncertainty while varying sample size. Supporting the result from Theorem 2, estimates converge as the sample size grows.

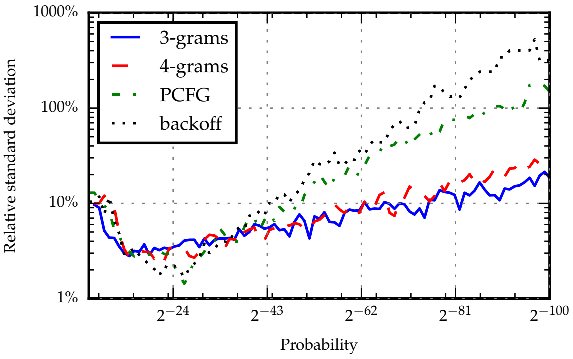


Figure 4: Estimation uncertainty for very small probabilities. When probabilities reach values for which elements in the sample are rare (see Figure 2), estimations lose precision.

ber of occurrences of a passwords falls below the backoff models' threshold, the PCFG model continues by guessing words that are in the training set, while the backoff model starts attempting "generalizations", attempting words that are not in the training set. At first, this favors PCFGs, but they eventually lose efficacy because many existing passwords are not represented by the patterns they model.

From Figure 7, we conclude that the optimal attack strategy varies depending on the number of guesses that the attacker can afford: PCFGs perform best at first but they are not capable of finding a substantial percentage of passwords, while 4-grams and then 3-grams become preferrable to attack stronger passwords. While never being optimal, the backoff strategy performs well across the whole spectrum of passwords, proving that it indeed "self-tunes" to avoid the dead-ends that models such as PCFGs reach.

In Figure 8, we analyze whether different attacks would require similar effort to guess the same password. In the scatter-plots, we compare the number of attempts needed to guess a password between the backoff strategy and the other ones considered in Figure 7. The correlation between the approaches is obvious, suggesting that the strength of a password against an attack is indicative of its strength against other attacks. In addition, we notice that the backoff and PCFG strategies are very highly correlated for the passwords that are easier to guess (up to around 2^{16} attempts), which are in general the ones that appear multiple times in the training dataset. When a password can ultimately be guessed by PCFGs, they do this with less efforts than the backoff attack; however, we stress that roughly a third of the passwords are never guessed by PCFGs and hence they do not appear in the graph.

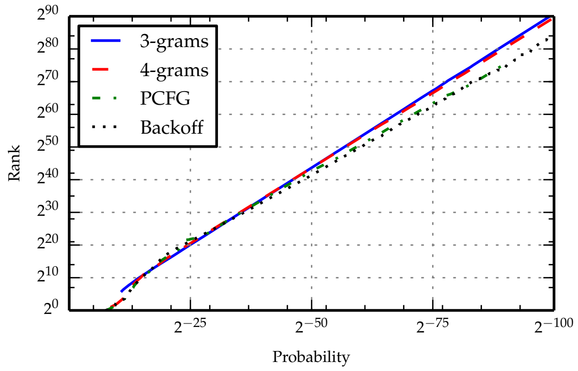


Figure 5: Probability vs. rank. Between n-grams and the backoff and PCFG models, the probabilities of passwords having the same rank can differ by orders of magnitude.

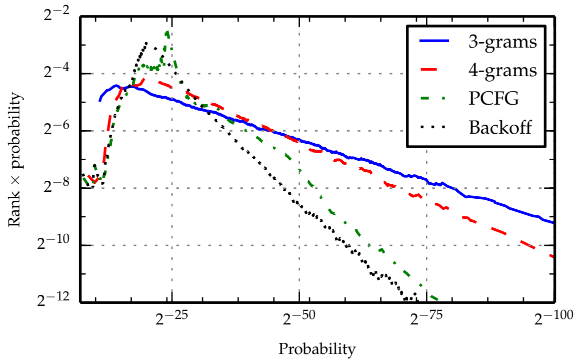


Figure 6: Probability multiplied by rank. Ma et al. [19] conjectured that the product between these two values remains between 2^{-3} and 2^{-8}. Our results show that this trend does not hold for small probabilities.

The results of Figure 8 highlight that the strength of a given password against different attacks is obviously correlated; nevertheless we consider that user passwords should still be checked against several attacks. A resonable conservative strategy would be to output to the user the result yielding the lowest strength value.

In the following, we provide more details about attacks by examining each class more in detail.

5.2.1 *Dictionary Attacks*

We take into account dictionary attacks. We consider the Openwall and dic-0294 dictionaries, two ad-hoc dictionaries for password guessing that have been evaluated in related work [7,19,30], and we compare them to using the password in the Xato dataset as a dictionary, sorted by decreasing number of occurrences: the results are shown in Figure 9. It appears that large password datasets make dictionaries obsolete: guessing passwords as they appear in the Xato dataset appears always preferable than using dictionaries. Using PCFGs is essentially equivalent to guessing passwords as they appear in the training dataset – however, passwords present in the Xato dataset are useful to find around 40% of the passwords of the Rockyou dataset; an additional 20% are found via the mangling made through PCFGs.

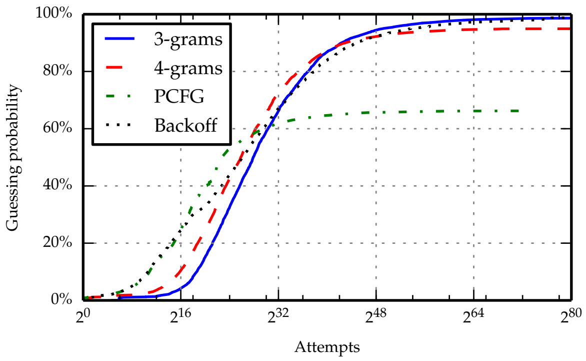


Figure 7: Overview of the best-performing attacks. Depending on the difficulty of the passwords, different attack strategies perform better. The “backoff” strategy perform not far from optimally for each probability.

5.2.2 *N-Grams*

In Figure 10, we compare attack models based on n-grams. Compared to results based on much smaller training sets [7], the sheer size of the training dataset makes it possible to find a large majority of the passwords even with higher-order models ($n = 3$ and $n = 4$). Attack models based on 2-grams and 1-grams (i.e., simply considering character frequencies) are unlikely to be helpful for conceivably-sized guessing attacks.

5.2.3 *PCFGs*

We analyze the performance of PCFG-based attacks in Figure 11. As described in Section 4.2.2, our implementation of the model allows us to perform “leave-one-out” cross-validation. In this case, we evaluate the benefits that an attacker might have with a very well tuned training set: the extreme case is, of course, to perform training on the *full* Rockyou database. As per the original description by Weir et al., we also train the alphabetic patterns of PCFGs from an external dictionary, rather than from the training set: we choose Openwall, since it performs best in Figure 9.

A better training set (Rockyou rather than Xato) boosts performance noticeably, raising the percentage of found passwords by around 13%. Two factors are in play here: first, the Rockyou training set is larger than Xato; second, it is – of course – more representative of passwords for the Rockyou website (e.g., the “rockyou” string is very common in the former and relatively rare in the latter). Again, we find that the size of training sets available today makes ad-hoc dictionaries obsolete, confirming previous findings [19].

5.2.4 *Backoff*

Figure 12 shows the results of evaluating different variants of the backoff model. Again, we evaluate the Rockyou dataset as a training set using the “leave-one-out” method. As before, we observe that using the matching dataset as a training set improves the attack. In addition, our modification to include a start symbol improves the quality of initial guesses, matching at the beginning the one of PCFGs (compare with Figure 7).

5.3 Impact of Training

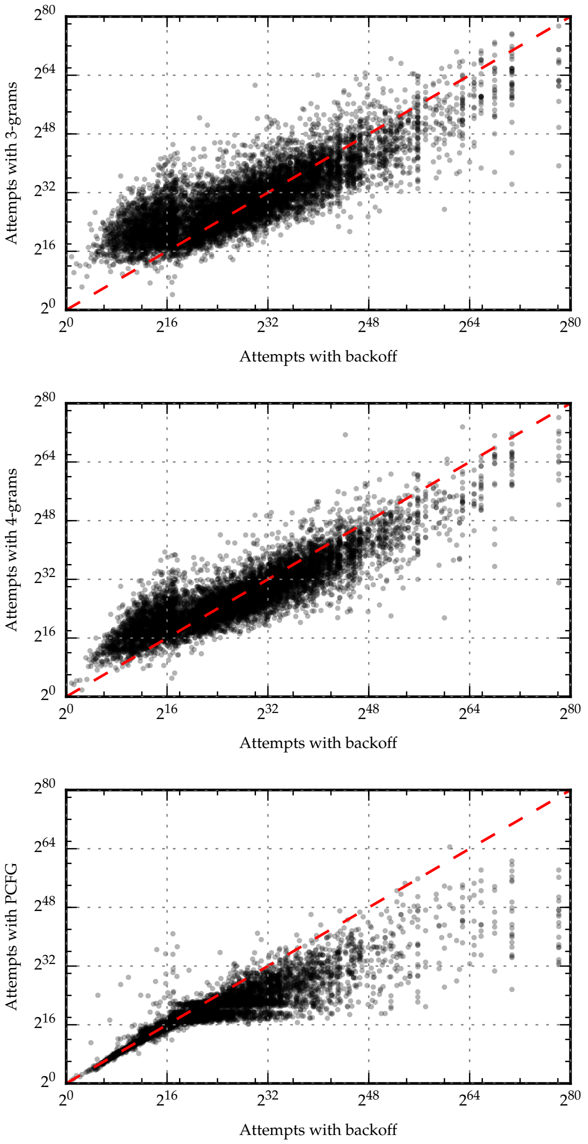


Figure 8: Comparing different attacks on the same passwords. Dots on the diagonal represents passwords that are found after the same number of attempts by two attacks.

In Figure 13, we consider the effect of training set size on our results: we consider the backoff model, because it obtains a good guessing probability for any number of attempts. We vary the training set size between 0.1% and 100% of the training dataset.

It is interesting to point out that a larger training set has little effect in guessing either the easiest passwords (the most common passwords are likely to be in smaller subsets of the original training set) or the hardest ones (they are most likely unique, and little information about them can be gleaned even from large training sets). Raising the training set size, instead, appears useful for the "average" passwords, i.e., those for which either the given password or a similar one can be found in a larger dataset.

5.4 Evaluating Mandatory Requirements

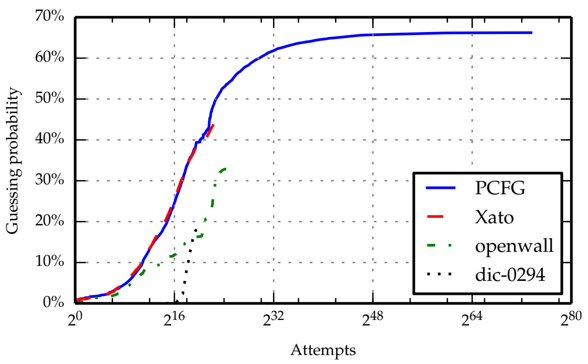


Figure 9: Dictionary attacks. A large sets of leaked passwords performs better in cracking passwords than ad-hoc crafted dictionaries. PCFGs are efficient to generalize guesses when dictionaries are exhausted.

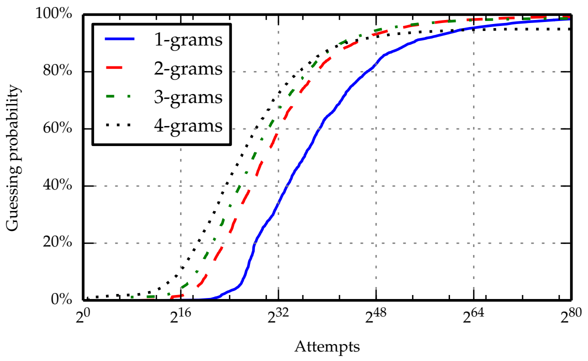


Figure 10: n-grams. When trained on very large datasets, attacks based on 3-grams cover almost all passwords. 4-grams are slightly more efficient to guess weaker passwords.

We now turn our attention to evaluating mandatory requirements such as the ones that are often included in passwords: minimum length and/or including uppercase, numeric and/or symbolic characters. From our datasets, we only consider the passwords that satisfy such requirements; we evaluate the number of attempts an attacker would need in order to guess a password. We assume that the attacker is aware of the restriction and therefore that passwords that do not satisfy the requirements will *not* be guessed.

We think that these results should be regarded as optimistic evaluations of the strength improvements that can be obtained by imposing such restrictions: indeed, users who choose weaker passwords might see such restrictions as a hurdle to bypass with minimum effort, choosing the simplest modification that would satisfy the requirement (e.g., appending "1" to satisfy the requirement of including digits).

In this case, given a boolean function $f(\alpha)$ that tells us whether a given password is accepted according to the requirements, our new strength measure should only count the passwords that have probability higher than $p(\beta) > p(\alpha)$ *and* for which $f(\beta)$ is true; the procedure to do this is described in Section 3.2. Passwords that satisfy the most stringent requirements are rare in our datasets, so to limit precision problems, we increase the size of our sample set

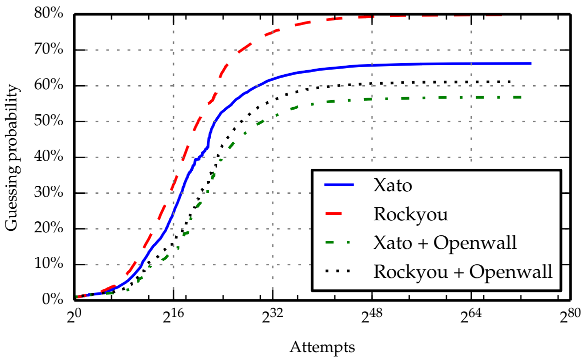


Figure 11: PCFGs. The legend indicates the training set and whether an additional dictionary has been used to train the alphabetic groups of PCFGs.

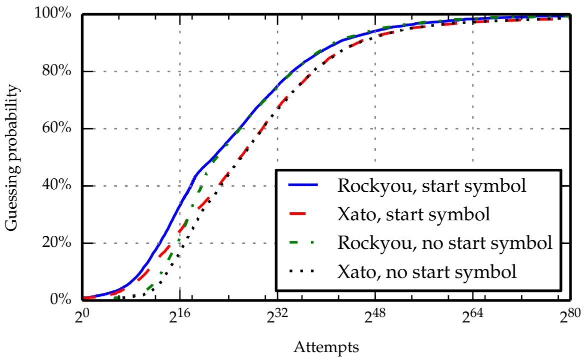


Figure 12: Backoff. The legend indicates the training set and whether our proposed use of a start symbol is adopted.

Θ and the test set from the Rockyou dataset, in order to obtain 10,000 passwords that satisfy f in both sets.

In Figure 14 we plot the distribution of password strength for passwords with restrictions, according to the procedure outlined above (policies are described in the caption). Again, we consider the backoff model as the attack method.

Length requirements increase password strength rather homogeneously for all passwords – with the exception of the weakest ones for the length limitation at 8: this is because several very common passwords have length 8.

We notice that requiring passwords with both letters and numbers does little to increase security; this may be due to the fact that passwords including letters and numbers often have very predictable patterns (e.g., numbers at the end) which are easily discovered by guessing techniques. Requiring a mix of uppercase and lowercase characters does better, but the strength improvement on the weakest passwords is limited: this is because, for weak passwords, uppercase characters are generally placed at the beginning of the password, and this is easy to guess for attackers. Symbols appear to be less predictable and placed in different locations of the password, and that noticeably increases password strength: in our case, passwords that include symbols are roughly as strong as those of length at least 12.

6. CONCLUSIONS

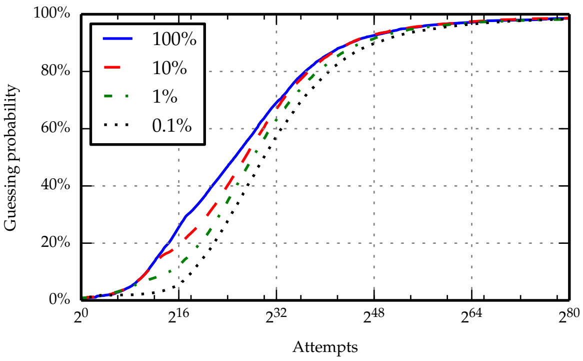


Figure 13: Training set. The legend refers to the percentage of the full Xato set used for training.

Passwords are an ubiquitous way of handling access control, and hence properly evaluating their solidity against state-of-the-art attacks is obviously important. To the best of our knowledge, this is the first study that provides a reliable estimation – backed by proofs of correctness and convergence – for the success rate of very expensive attacks to passwords using state-of-the-art guessing techniques. By evaluating the password strength in terms of number of guesses needed by an attacker, we make it possible to gain better insights into the actual cost that an attacker should pay to guess a password, and therefore better understand the usability vs. security trade-off that password strength implies.

Our study shows that the number of attempts needed to guess a password is correlated between different attacks; nevertheless, we believe that user passwords should be checked against several attack models, and that a reasonable strategy would be to conservatively output the result yielding the lowest strength value.

Our method is generic and extremely lightweight, with the possibility to increase the level of accuracy in the assessment of password strength by a quantity that can be directly related to the number of computations that one is willing to invest. Our proposal is applicable to *any* generative probabilistic model; we consider this approach to be reasonably future-proof, as we believe that new developments in password guessing will most likely continue to use models that are (or can be expressed as) generative probabilistic models.

7. REFERENCES

[1] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *S&P*. IEEE, 2012.

[2] G. Bontempi and S. Ben Taieb. *Statistical foundations of machine learning*. 2009.

[3] M. Burnett. Today I am releasing ten million passwords. `https://xato.net/passwords/ten-million-passwords/`, February 2015.

[4] W. Burr, D. Dodson, R. Perlner, W. Polk, and S. Gupta. NIST special publication 800-63-1 electronic authentication guideline, 2006.

[5] C. Castelluccia, M. Dürmuth, and D. Perito. Adaptive password-strength meters from markov models. In *NDSS*. Internet Society, 2012.

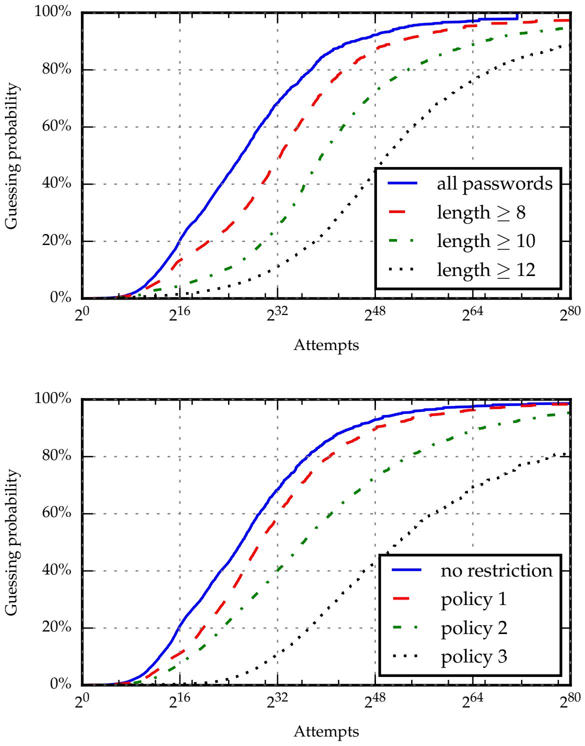


Figure 14: Restrictions. We consider the number of attempts an attacker would need to guess passwords that satisfy certain restrictions, when knowing that the restrictions are present. Explanation of the labels: “policy 1” refers to passwords that have alphabetic and numeric characters; “policy 2” requires lowercase, uppercase and digits; “policy 3” requires alphabetic and numeric characters in addition to non-alphanumeric symbols.

[6] X. de Carné de Carnavalet and M. Mannan. From very weak to very strong: Analyzing password-strength meters. In *NDSS*. Internet Society, 2014.

[7] M. Dell’Amico, P. Michiardi, and Y. Roudier. Password strength: An empirical analysis. In *INFOCOM*. IEEE, 2010.

[8] M. Duermuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane. OMEN: Faster password guessing using an ordered Markov enumerator. In *ESSoS*. IEEE, 2015.

[9] 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 *SIGCHI*. ACM, 2013.

[10] D. Florêncio, C. Herley, and P. C. Van Oorschot. An administrator’s guide to internet password research. In *LISA*. USENIX, 2014.

[11] D. Florêncio, C. Herley, and P. C. Van Oorschot. Password portfolios and the finite-effort user: Sustainably managing large numbers of accounts. In *USENIX Security*, 2014.

[12] C. Herley and P. Van Oorschot. A research agenda acknowledging the persistence of passwords. In *S&P*. IEEE, 2012.

[13] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *J. Am. Stat. Assoc.*, 47(260):663–685, 1952.

[14] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE TASSP*, 35(3):400–401, 1987.

[15] 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 *S&P*. IEEE, 2012.

[16] D. V. Klein. Foiling the cracker: A survey of, and improvements to, password security. In *USENIX Security*, 1990.

[17] S. Komanduri, R. Shay, L. F. Cranor, C. Herley, and S. Schechter. Telepathwords: Preventing weak passwords by reading users’ minds. In *USENIX Security*, 2014.

[18] Z. Li, W. Han, and W. Xu. A large-scale empirical analysis of chinese web passwords. In *USENIX Security*, 2014.

[19] J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *S&P*. IEEE, 2014.

[20] R. Morris and K. Thompson. Password security: A case history. *CACM*, 22(11):594–597, 1979.

[21] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *CCS*. ACM, 2005.

[22] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO*. Springer, 2003.

[23] C. Percival and S. Josefsson. The scrypt password-based key derivation function. 2012.

[24] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[25] Solar Designer and S. Marechal. Password security: past, present, future. Password^12 workshop, December 2012.

[26] E. H. Spafford. Observing reusable password choices. In *USENIX Security*, 1992.

[27] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. 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 *USENIX Security*, 2012.

[28] R. Veras, C. Collins, and J. Thorpe. On the semantic patterns of passwords and their security impact. In *NDSS*. Internet Society, 2014.

[29] M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *CCS*. ACM, 2010.

[30] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *S&P*. IEEE, 2009.