An Explainable Online Password Strength Estimator

Liron David(✉) and Avishai Wool(✉)

School of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel
`lirondavid@gmail.com`, `yash@eng.tau.ac.il`

Abstract. Human-chosen passwords are the dominant form of authentication systems. Passwords strength estimators are used to help users avoid picking weak passwords by predicting how many attempts a password cracker would need until it finds a given password.

In this paper we propose a novel password strength estimator, called PESrank, which accurately models the behavior of a powerful password cracker. PESrank calculates the rank of a given password in an optimal descending order of likelihood. PESrank estimates a given password's rank in fractions of a second—without actually enumerating the passwords—so it is practical for online use. It also has a training time that is drastically shorter than previous methods. Moreover, PESrank is efficiently *tweakable* to allow model personalization in fractions of a second, without the need to retrain the model; and it is *explainable*: it is able to provide information on *why* the password has its calculated rank, and gives the user insight on how to pick a better password.

We implemented PESrank in Python and conducted an extensive evaluation study of it. We also integrated it into the registration page of a course at our university. Even with a model based on 905 million passwords, the response time was well under 1 s, with up to a 1-bit accuracy margin between the upper bound and the lower bound on the rank.

1 Introduction

1.1 Background

Text passwords are still the most popular authentication and are still in widespread use specially for online authentication on the Internet. Unfortunately, users often choose predictable and easy passwords, enabling password guessing attacks. Password strength estimators are used to help users avoid picking weak passwords. Usually they appear as password meters that provide visual feedback on password strength [42]. The most precise definition of password's strength is *the number of attempts that an attacker would need in order to guess it* [13].

A common way to evaluate the strength of a password is by heuristic methods, e.g., based on counts of lower- and uppercase characters, digits, and symbols (LUDS). Theses password-composition policies have grown increasingly complex [25]. Despite it being well-known that these do not accurately capture password strength [46], they are still used in practice.

E. Bertino et al. (Eds.): ESORICS 2021, LNCS 12972, pp. 285–304, 2021.
https://doi.org/10.1007/978-3-030-88418-5_14

Subsequently, more sophisticated, cracker-based, password strength estimators have been proposed. In a cracker-based estimator, either an actual password cracker is utilized to evaluate the password strength—or the estimator uses an accurate model of the number of attempts a particular cracker would use until reaching the given password. The main approaches have been based on, e.g., Markov models [15,31,35], probabilistic context-free grammars (PCFGs) [28,47], neural networks [34,41] and others [20,49]. Based on the well known phenomenon that people often use attributes of their personal information in their passwords (their names, email addresses, birthdays etc.), [23,29,45] have proposed to tweak the prior models, based on personal information known about a given user's password.

In this work we propose a novel addition to this line of research, called PESrank. Our goal is to provide a password strength estimator that enjoys the following properties:

- It is a cracker-based estimator, that accurately models the behavior of a powerful password cracker. The modeled cracker calculates the rank of a given password in an optimal descending order of likelihood.
- It is practical for *online* use, and is able to estimate a given password's rank in fractions of a second—i.e., without actually enumerating the passwords.
- Has reasonable training time, drastically shorter than previous methods (some of which require days of training).
- It is efficiently *tweakable* to allow model personalization, without the need to retrain the model.
- It is *explainable*, and provides feedback on *why* the password has its calculated rank, giving the user insight on how to pick a better password.

1.2 Contributions

Our idea in the design of PESrank is to cast the question of password rank estimation in a probabilistic framework used in side-channel cryptanalysis. We view each password as a point in a d-dimensional search space, and learn the probability distribution of each dimension separately. This learning process is based on empirical password frequencies extracted from leaked password corpora, that are projected onto the d dimensions. Once the d probability distributions are learned, the a-priori probability of a given password is the *product* of the d probabilities of its sub-passwords.

Using this model, optimal-order password cracking is done by searching the space in decreasing order of a-priori password probability, which is analogous to side-channel key enumeration; likewise, password strength estimation is analogous to side-channel rank estimation. There is extensive research and well known algorithms for both problems in the side-channel cryptanalysis literature. We adopt a leading side-channel rank estimation (ESrank, [10]) for use in PESrank—which accurately models an optimal key enumeration, or equivalently, password enumeration algorithm. The ESrank algorithm also has accuracy guarantees providing both upper-and lower-bounds on the true rank.

PESrank's training time is very reasonable, taking minutes-to-hours. It is also efficiently *tweakable* to allow model personalization, without the need to retrain the model. In addition, PESrank is able to *explain* the password strength value: For example, PESrank can indicate that the password newyork123 is based on a leaked word that was used by 129,023 people, and uses a very popular suffix that was used by over 17 million people. Such explainability is very important since it helps guide the user on how to pick better passwords. This is in contrast to prior methods, especially those based on neural networks, which offer little - to no explainability.

In order to demonstrate PESrank's capabilities as an online password strength estimator we implemented PESrank in Python and integrated it into the registration page of a course at our university. Even with a model based on 905 million passwords, the end-to-end response time in the browser was well under 1 s, with up to a 1-bit accuracy margin between the upper bound and the lower bound on the rank. This allowed us to run a proof-of-concept study on how students reacted to their passwords' strength estimates.

We conducted an extensive evaluation study comparing PESrank's accuracy to prior approaches. In our study we used Ur et al.'s Password Guessability Service [5] (PGS), which provides access to the Hashcat [21] and John the Ripper [36] crackers, the Markov [35] and PCFGs [47] methods, and to the neural-network method [34] (Monte-Carlo variant). We compared the ranks calculated by PESrank to the ranks obtained by these five password strength estimators. We show that PESrank (and, in fact, the optimal password cracker it models) is more powerful than previous methods: the model-based cracker can crack more passwords, with fewer attempts, than the password crackers we compared it to for crackable passwords whose rank is smaller than 10^{11}. Due to space constrains, many details have been omitted from this paper and are present in our full technical report [2].

2 Rank Estimation and Key Enumeration in Cryptographic Side-Channel Attacks

Side-channel attacks (SCA) represent a serious threat to the security of cryptographic hardware products. As such, they reveal the secret key of a cryptosystem based on leakage information gained from physical implementation of the cryptosystem on different devices. Information provided by sources such as timing [27], power consumption [26], electromagnetic emulation [39], electromagnetic radiation [1,16] and other sources, can be exploited by SCA to break cryptosystems. A security evaluation of a cryptographic device should determine whether an implementation is secure against such an attack. To do so, the evaluator needs to determine how much time, what kind of computing power and how much storage a malicious attacker would need to recover the key given the side-channel leakages. The leakage of cryptographic implementations is highly device-specific, therefore the usual strategy for an evaluation laboratory is to launch a set of popular attacks, and to determine whether the adversary can break the implementation (i.e., recover the key) using "reasonable" efforts.

Most of the attacks that have been published in the literature are based on a "divide-and-conquer" strategy. In the first "divide" part, the cryptanalyst recovers multi-dimensional information about different parts of the key, usually called subkeys (e.g., each of the $d = 16$ AES key bytes can be a subkey). In the "conquer" part the cryptanalyst combines the information all together in an efficient way via key enumeration, for one of two purposes as follows.

The Key Enumeration Problem. The cryptanalyst obtains d independent subkey spaces $k_1, ..., k_d$, each of size n, and their corresponding probability distributions $P_{k_1}, ..., P_{k_d}$. The problem is to enumerate the full-key space in decreasing probability order, from the most likely key to the least, when the probability of a full key is defined as the product of its subkey's probabilities, and test each full key in turn until the correct secret key is found.

A naive solution for key enumeration is to take the Cartesian product of the d dimensions, and sort the n^d full keys in decreasing order of probability. However this approach is generally infeasible due to both time and space complexity. Therefore several algorithms offering better time/space tradeoffs have been devised. The currently best optimal-order key enumeration is [44], with an $O(n^{d/2})$ space complexity, and near-optimal-order key enumeration algorithms with drastically lower space complexities are those of [3,9,32,33,38].

Unlike a cryptanalyst trying to extract the secret key, a security evaluator knows the secret key and aims to estimate the number of decryption attempts the attacker needs to do before he reaches the correct key, assuming the attacker uses the SCA's multi-dimensional probability distributions. Formally:

The Rank Estimation Problem: Given d independent subkey spaces of sizes n_i for $i = 1, \ldots, d$ with their corresponding probability distributions $P_1, ..., P_d$ such that P_i is sorted in decreasing order of probabilities, and given a key k^* indexed by $(k_1, ..., k_d)$, let $p^* = P_1(k_1) \cdot P_2(k_2) \cdot ... \cdot P_d(k_d)$ be the probability of k^* to be the correct key. The problem is to estimate the number of full keys with probability higher than p^*, when the probability of a full key is defined as the product of its subkey's probabilities. In other words, the evaluator would like to estimate k^*'s *rank*: the position of the key k^* in the list of n^d possible keys when the list is sorted in decreasing probability order, from the most likely key to the least.

While enumerating the keys in the optimal SCA-predicted order is a correct strategy for the evaluator, it is limited by the computational power of the evaluator. Hence using algorithms to estimate the rank of a given key, without enumeration, is of great interest. Multiple rank estimation algorithms appear in the literature, the best of which are currently [10,17,33]. They all work in fractions of a second and generally offer sub 1-bit accuracy (so up to a multiplicative factor of 2).

3 Multi-dimensional Models for Passwords

3.1 Overview

The starting point in producing a password strength estimator is a leaked password corpus. The frequency of appearance of each leaked password provides an a-priori probability distribution over the leaked passwords. Given a hash of an unknown password, trying the leaked passwords in decreasing frequency order, is the optimal strategy for a password cracker—if the password at hand is in the corpus. To crack passwords that are not in the leaked corpus *as-is*, password crackers rely on the observation that people often take a word, which we shall call the *base word*, and mutate it using various transformations such as adding digits and symbols before or after the base word, capitalizing some of the base word's letters, or replacing letters by digits or symbols that are visually similar using "l33t" translations.

Our main idea is that if we can represent the list of base words as a dimension, and represent each possible class of transformations as another independent dimension, we can pose the password cracking problem as a key enumeration problem, and similarly, pose the password strength estimation as a rank estimation problem. Each dimension should have its own probability distribution. Once we pose the password strength estimation question this way, we can use existing algorithms. A multi-dimensional password cracker would enumerate combinations of base word plus a transformation in every dimension, in decreasing order of the *product* of per-dimension a-priori probabilities. For each combination it would apply the current set of transformations to the base word, and test the password. The matching multi-dimensional password strength estimator decomposes a given password into its base word and a transformation in every dimension, uses the model to calculate the a-priori probability of the password, and then estimates its rank *without enumeration.*

Thus, we arrive at the following framework: First, identify meaningful classes of transformations, and find a suitable representation for each as a dimension. Next, build a probability distribution for each dimension using the training corpus, to create a model. Finally, use a good rank estimation algorithm with the model and evaluate its performance.

3.2 The Data Corpus

To study the statistical properties of passwords, and then to train our method, we used Jason's corpus of leaked passwords [24]. This corpus contains 1.4 billion pairs of username and password, compiled from multiple leaked corpora: Yahoo, Target, Facebook, Hotmail, Twitter, MySpace, hacked PHPBB instances, and many other sources. We believe that Jason's corpus is a superset of the corpora used to train previous methods. After eliminating passwords that contain non-ASCII characters and eliminated garbage "passwords" of more than 32 characters we obtained a corpus of 905,060,363 passwords.

Table 1. Leaked password patterns in Jason's corpus

Start with digits/symbols	8.946%
End with digits/symbols	50.237%
Use capital letters	7.665%
Use l33t transformations	9.863%

From this corpus we sampled 300,000 username-password pairs, to serve as a test set. We split the test set into 10 separate samples, of 30,000 passwords each, and submitted all the sample sets to PGS for evaluation. To compare the ranks we received from PGS to those of PESrank, we trained PESrank on the same training corpora used by the PGS implementations of the various methods, as follows:

- **PGS set:** According to [5] the PCFG, Markov, hashcat and JtR algorithms were trained on 6 corpora, totalling 33 million passwords, plus 6 million natural language words, collectively called the "PGS training set". We used this set to select the dimensions of PESrank and when we compared the performance of PESrank to that of PCFG, Markov, hashcat and JtR.
- **PGS++ set:** According to [5] the Neural algorithm was trained on a passwords from a large superset of the PGS set including 26 additional corpora, called the "PGS++ training set". We used this set when comparing to the Neural algorithm.
- **Jason:** To test PESrank's training time, for the usability proof of concept and for stand-along performance evaluation we used the full Jason corpus with its 905 million leaked passwords.

3.3 Selecting Dimensions

Following [49] we chose the dimensions according to the patterns humans tend to choose in their passwords: prefixes and suffixes (e.g., iloveyou!! or 123iloveyou), mixed letter case (e.g., iLoVeyOu), and leet speak (e.g., il0v3you). We also verified the observations of [49] in Jason's corpus: Table 1 shows that significant fractions of the leaked passwords fit our choice of dimensions. After checking several options, (see technical report [2]), we chose the following five dimensions: prefix, base word, suffix, capitalization and l33t. Next we describe each dimension separately:

We define "prefix" as the string consisting of all the digits and symbols that appear to the left of the leftmost letter of the password, and define "suffix" as the string consisting of the digits and symbols that appear to the right of the rightmost letter in the password. We define "base word" as the string starting with the leftmost letter and ending with the rightmost letter of the password. For example, if password is '123Pa$$w0rd!!', the prefix is '123', the suffix is '!!' and the base word is 'Pa$$w0rd'. The base word can consist of mixed-case letters, digits and/or symbols. In case there are no letters in the password, (e.g., '1234567890'), the password itself is considered to be the base word, and the prefix and suffix are the empty strings. In case the password starts with a letter, (e.g., 'abc123'), the prefix is the empty string, and similarly, if the password ends with a letter, ('123abc'), the suffix is the empty string. Note that the division into these three parts is purely syntactical and is not a semantic division as in [43]. Computing a good semantic division is time and space consuming, so we elected to rely on the simpler syntax-based division.

Next we define "capitalization pattern" as the list of positions of capital letters in the base word. In order to decrease the dependency between the password length and the capitalization pattern, we elected to represent the capitalization pattern as a list of positive and negative indices at which capital letters appear: The negative indices count from the end of the base word, and the positive indices count from the base word start. To avoid ambiguity, both the negative and the positive indices do not exceed the middle index. We also added a capitalization pattern 'all-cap' for the special case in which all the letters are capitals (regardless of password length). If there is no capital letter in the base word, the capitalization pattern is empty.

Note that the capitalization-pattern dimension is not strictly independent of the base-word dimension: e.g., a capitalization pattern t may refer to indices that are outside a short base word b, or b's characters at the indexed positions may be symbols or digits (which do not have a capitalized form). In such cases the transformation t degenerates into the null transformation. For a model-based password cracker, this dependence implies some inefficiency, since the cracker will test the same password multiple times, once for each capitalization-pattern that is equivalent to the null transformation for the current base word. The rank estimation accurately accounts for such a cracker's inefficiency. This means that a more sophisticated cracker can be developed: it could skip null transformations and save itself time. Thus this dependency only increases the password strength estimation value, which is still, as we see in Sect. 5, better and lower than existing methods' estimation.

For example for '123Pa$$w0rD!!' the capitalization pattern of the base word 'Pa$$w0rD' is the first letter and the last letter, denoted by '[0,−1]', and for the base word 'PASSWORD' the capitalization pattern is 'all-cap'.

Finally, we define "l33t pattern" as a list of l33t transformation in the base word. The l33t pattern depends on the position of the letter being mutated, and on the choice of replacement (Table 2 shows that some letters have more than one l33t replacement). We elected to ignore the positionality aspect. We numbered the possible l33t replacements from 1 to 14—e.g., transforming 'a' into '4' is transformation number 3—and represent the whole l33t transformation of a base word by a tuple of l33t replacement numbers. So for '123Pa$$w0rD!!' the l33t transformation is s$\leftrightarrow$ \$ and o $\leftrightarrow$ 0. We assume that if a l33t replacement is applied then it is applied to *all* the relevant letters in the base word. So following Table 2, the l33t pattern of '123Pa$$w0rD!!' is '[1,4]' which means "replace all occurrences of o by 0 and all occurrences of s by \$". If there are no l33t transformations in the base word, the base word remains as-is and the

Table 2. L33t transformations

Index	1	2,3	4,5	6	7,8	9,10	11	12,13	14
Letter	o	a	s	e	g	t	z	i	x
l33t	0	[@,4]	[$,5]	3	[6,9]	[+,7]	2	[1,!]	%

l33t pattern is empty. Note that the l33t-pattern dimension is not independent of the base-word dimension, and as before, this dependency introduces some inefficiency to a model-based password cracker.

3.4 The Learning Phase

We learn the distributions of the prefix, base word, suffix, capitalization and l33t using the training set at hand—recall Sect. 3.2—as follows. Let these distributions be denoted by P_1, P_2, P_3, P_4, P_5 respectively. For each password in the training set, we divide the password into its five sub-passwords, as described above, and increment the dimensional-frequency of each prefix/suffix/base word/capitalization/l33t sub-password by 1. Before incrementing the base word's frequency, we "uncapitalize" and "unl33t" it, i.e., we ensure that all the base word letters are in lower case having no l33t transformation, so for the raw base word 'Pa$$w0rD' we increment the frequency of 'password' in the base word dimension P_2. Finally we normalize the five lists of frequencies into probability distributions, and sort them in decreasing order.

Following [14,30,40], we know that people have a tendency to choose passwords that contain dates and meaningful numbers. To take this observation into account, we enriched the probability distributions of the prefix, base word, and suffix dimensions (before normalizing), by adding the following strings that may not present in the training corpus: (1) All the digit sequences of up to 6 digits were added to the prefix and suffix distributions. (2) All the digit sequences of length exactly 6 were added to the base word distribution. Each extra sub-password was added with a frequency $\epsilon = 0.5$ to account for the fact that it didn't appear in the corpus.

3.5 The Estimation Phase

A model-based cracker based on, e.g., [47] goes over the password candidates using an optimal-order enumeration. We can use a matching rank estimation algorithm such as ESrank [10] to estimate the password guessability. Given a password P, we split into its sub-passwords $P = p^*||b^*||s^*||c^*||l^*$ where p^* is a prefix, b^* is a base word, s^* is a suffix, c^* is a capitalization and l^* is a l33t. With this, using the five probability distributions P_1, P_2, P_3, P_4, P_5, we can apply a rank estimation algorithm such as [10]. The algorithm estimates the number of 5-part passwords $p_i||b_j||s_k||c_w||l_t$ (split in the same way), whose probabilities obey

$$P_1(i) \cdot P_2(j) \cdot P_3(k) \cdot P_4(w) \cdot P_5(t) \geq P_1(p^*) \cdot P_2(b^*) \cdot P_3(s^*) \cdot P_4(c^*) \cdot P_5(l^*).$$

In other words, it estimates the number of guesses a model-based cracker would attempt before reaching the given password P.

For a given password which is composed only of digits, the model may include several options to reach this password by the model-based password cracker since a numeric password can be divided into prefix, base word, and suffix, in different ways. To account for this condition in the rank estimation, we added special handling of numeric passwords. For such a password, the PESrank algorithm iterates over all its possible divisions into 3 sub-passwords (of any length): for an ℓ-digit password there are exactly $(\ell+1)(\ell+2)/2$ possibilities. For each division whose 3 sub-passwords appear in the model we calculate the password's probability. Finally, we return the rank of the division with the highest probability, since this is the division that will be encountered first by the optimal enumeration algorithm.

3.6 Estimating the Ranks of Unleaked Password Parts

As described so far, if even one of a given password's five parts is not present in the relevant dimension, PESrank is unable to estimate its rank. Following Komanduri [28], we also introduce an optional "unleaked" smoothing mode to PESrank: when it's active, the model is also able to provide an estimate of the strength of passwords with unleaked parts, for which we have no empirical a-priori probability. We do this as follows: if the password part s is missing from distribution P_i, that dimension's contribution to the password's probability is taken as $\alpha P_i(n_i)$: i.e., we use the probability of the least likely value in dimension i multiplied by an arbitrary fraction $\alpha < 1$.

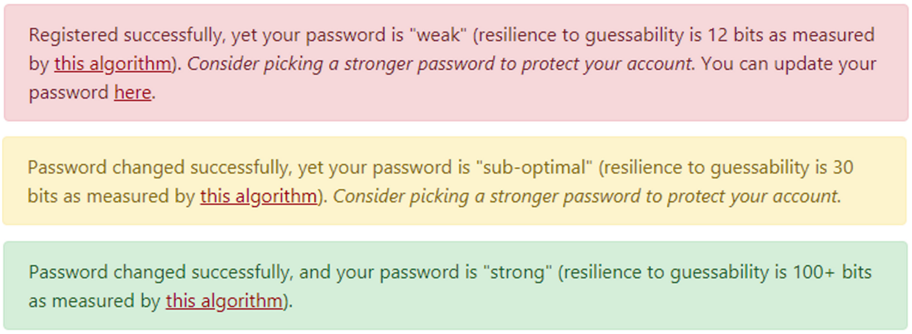
3.7 Performance

We tested our Python implementation of PESrank's training on a 3.40GHz core 7 PC running Windows 8.1 64-bit with 32GB RAM. The PESrank code is publicity available at GitHub [11]. We found that the PESrank training phase is quite fast—much faster than reported for previous methods. It takes only 12 min to train PESrank on the PGS set, in comparison to the days of training reported for the Markov [31] or PCFG [47] methods using the same set. To train our method on the PGS++ set, it took only 32 min, in comparison to the days it took to train the neural method [34] on the same data (see more details in Sect. 5). Because the PESrank training time is fast, we are able to train PESrank on the Jason corpus with 905 million passwords (an order of magnitude larger than the PGS++ set), and even on this corpus the training only took 4.5 h. The results are summarized in Table 3.

The table shows that on average an estimation takes 33 ms, and under 1 s in all cases, giving a good user experience. The lookup time includes: (1) dividing the password to its five dimensions' values, $v_1, \ldots, v_5$ (2) applying binary search for each dimension value v_i in its corresponding probability list P_i to obtain its probability $\Pr(v_i)$ (3) calculating the password probability $p = \prod_{i=1}^{5} \Pr(v_i)$ (4)

Table 3. PESrank performance metrics.

	PGS	Jason
Training time	12 min	4.5 h
Total space	660 MB	7.69 GB
Average estimation time per password	0.024 s	0.033 s
Maximum estimation time per password	0.690 s	0.792 s
Combined length of the two merged lists	768 integers	884 integers


Registered successfully, yet your password is "weak" (resilience to guessability is 12 bits as measured by this algorithm). *Consider picking a stronger password to protect your account.* You can update your password here.

Password changed successfully, yet your password is "sub-optimal" (resilience to guessability is 30 bits as measured by this algorithm). *Consider picking a stronger password to protect your account.*

Password changed successfully, and your password is "strong" (resilience to guessability is 100+ bits as measured by this algorithm).

Fig. 1. The possible messages shown by the registration page.

applying the ESrank algorithm [10] to find the rank r of the given probability p. The password's strength estimation is measured in bits: $\log_2(r)$.

4 Usability of PESrank

4.1 A Proof of Concept Study

We integrated PESrank into the registration page of the Infosec course. The updated system provides users with a gentle "nudge": it accepts weak passwords, yet tells the owners they are weak, and makes it easy for them to try again. The system displays three different messages, see Fig. 1: passwords with strength below 30 bits are considered 'weak' (red), strengths between 30–50 bits are considered 'sub-optimal' (yellow) and strengths above 50 bits are considered 'strong' (green). During the registration we only saved the password strength and *not the password itself* for statistical analysis, as approved by university's ethics review board. The total time from clicking on the Register button until the browser shows the feedback message (including password registration, strength estimation, network delays, and browser rendering) is well under 1 s. The increase in registration time due to the strength estimation was negligible and qualitatively unnoticeable.

There were 98 students who registered to this course: The median password strength of the first password chosen by the students was 41.51 bits, with the

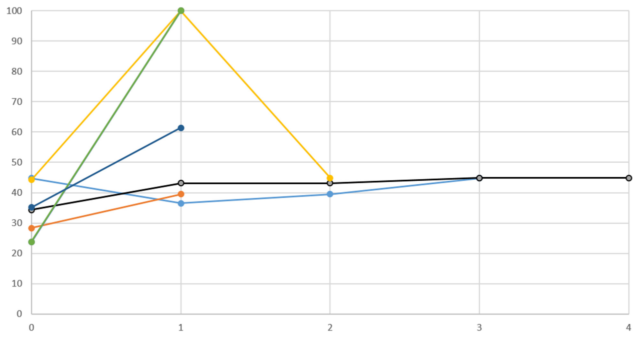
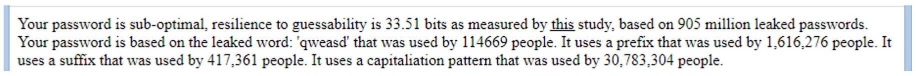


Fig. 2. The password strength versus password-change number for the 7 students who changed their password: index 0 indicates the strength of the initial password chosen by each student.



Your password is sub-optimal, resilience to guessability is 33.51 bits as measured by this study, based on 905 million leaked passwords. Your password is based on the leaked word: 'qweasd' that was used by 114669 people. It uses a prefix that was used by 1,616,276 people. It uses a suffix that was used by 417,361 people. It uses a capitaliation pattern that was used by 30,783,304 people.

Fig. 3. The PESrank implementation as a Google Cloud Function

weakest having strength of 14.14 bits. Out of the 98 students, 7 students changed their passwords to stronger passwords. The median strength of these students' first passwords was 34.32 bits, and the median strength of their final passwords was 44.88 bits: a significant improvement. In Fig. 2 we can see the evolution of passwords strengths of the seven students who changed their password (there are two students whose lines overlap due to similar strength choices). The figure shows that 5 students indeed picked a stronger password in their first change—one of whom later changed the password a second time in favor of a weaker password. Interestingly, two students changed their passwords 3 and 4 times, respectively, without significantly improving their strength.

4.2 Explainability

The anecdotal evidence from the proof of concept leads us to realize that while providing the password strength encourages some users to pick a better password, a good strength estimator should give the user guidance on *how* to pick a better password. One of the advantages of PESrank is that it is inherently very "explainable". As part of its calculation it discovers the a-priority probability (and frequency) of each sub-password - and this information can be shown to the user. E.g., in the latest version of the code, for the password NewY0rk123 we provide the following feedback: "Your password is sub-optimal, its guessability strength is 32 bits, based on 905 million leaked passwords. Your password is based on the leaked word: 'newyork' that was used by 129,023 people. It uses a suffix that was used by 17,631,940 people. It uses a capitalization pattern that was used by 592,568 people. It uses a l33t pattern that was used by 4,395,598 people".

This tells the user that (a) the transformations do not hide the leaked base word, that (b) they use a very common suffix and that (c) a simple l33t transformation is only marginally effective. And most importantly - it teaches that the split into the five dimensions is something password crackers know about and take advantage of. Furthermore, if the password has parts that are unleaked, this means the user actually selected a strong password part in that dimension—and PESrank is able to explain this. E.g. on a password "Dmmihhvk123", it would explain "Your password is strong, its guessability strength is 53 bits, based on 905 million leaked passwords. Your password is based on a good (unleaked) base word. It uses a suffix that was used by 17,631,940 people and a capitalization pattern used by 34,102,338 people". See Fig. 3 for a screenshot of the Google Cloud Function implementation. We are planning to conduct a wider scale experiment in the future using the improved code.

5 Comparison with Existing Methods

In order to test the power and accuracy of PESrank, we compared it to five cracker-based methods offered in PGS [5]: (1) the Markov model [31]; (2) the PCFGs model [47] with Komanduri's improvements [28]; (3) Hashcat [21]; (4) John the Ripper [36] mangled dictionary models. Then we compared it to the neural network-based model of Melicher et al. (with Monte-Carlo estimation) [34]. In all cases we used the algorithms' default settings and training data. To have a fair comparison, we trained PESrank on the PGS set for comparison with the PCFG, Markov, hashcat and JtR algorithms, and on the PGS++ set for comparison with the Neural algorithm. We also trained PESrank method on the full Jason corpus with its 905 million passwords in order to see how the training set size affects performance.

In order to evaluate PESrank and compare its performance to existing methods we computed for each model the percentage of passwords that would be cracked after a particular number of guesses: in other words, the Cumulative Distribution Function (CDF) of each model. More powerful guessing methods guess a higher percentage of passwords in our test set, and do so with fewer guesses: hence a better model has a CDF that rises more sharply and ultimately reaches a higher percentage.

When "unleaked" smoothing mode is active (recall Sect. 3.6), the CDF exhibits two features: first, the CDF has a sharp upward "jump" at a rank that is influenced by the choice of α, and second, the CDF by definition always reaches 100%: even a password that has unleaked parts in all 5 dimensions will receive a minimal probability of $\alpha^5 \cdot \prod_{i=1}^{5} P_i(n_i)$, which will translate to a rank equal to the models volume $\prod_{i=1}^{5} n_i$. In order to emphasize the arbitrary nature of the"unleaked" parts of the CDF, when we show figures created in "unleaked" smoothing mode, the relevant parts of the CDF are marked in a dotted line.

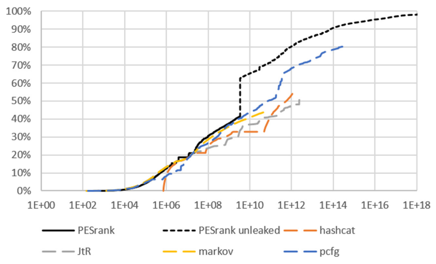


Fig. 4. The CDFs of PESrank in unleaked mode versus PCFG, Markov, Hashcat and JtR

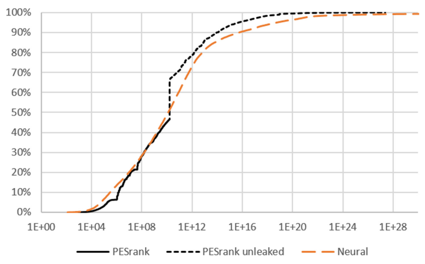


Fig. 5. The CDFs of PESrank in unleaked mode versus Neural.

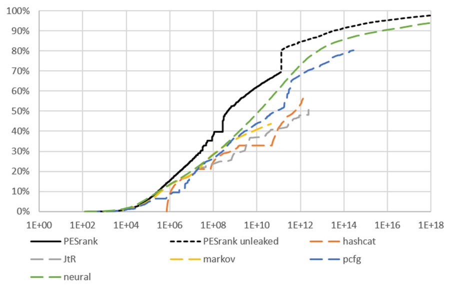


Fig. 6. The CDF of each method trained on all the passwords available to it: PESrank in unleaked mode - on Jason, Neural - on PGS++, and the rest - on PGS.

5.1 Comparison to Cracker-Based and Neural Methods

Figure 4 shows the comparing PESrank, trained on the PGS set, with PCFG, Markov, hashcat and JtR algorithms with PESrank in "unleaked" mode, with $\alpha = 10^{-3}$: note the jump around 10^9 and the dotted curve beyond it, indicating that higher ranks rely on password parts for which there is no empirical a-priori probability. We note that while PCFG does use Komanduri's smoothing [28], there is no external indication when this extrapolating calculation is applied: we speculate that it may be occurring around rank 10^{11}, where an upward "jump" can be observed in the PCFG curve.

Figure 5 shows the results comparing PESrank, in "unleaked" mode, trained on the PGS++ set, with the Neural method (with Monte Carlo estimation). We see that PESrank is on-par with the Neural method for "practically crackable" passwords (up to the "unleaked" mode jump where the dotted curve begins).

Figure 6 shows the CDFs of all the methods we compared, each trained on the maximal training set available to it (PESrank in "unleaked" mode). When PESrank is trained on a 905 million password training set, its advantage over the other methods, as provided by the PGS service [5], grows. While this figure mostly demonstrates the advantage of using a larger training set, it also shows that PESrank is actually able to digest such a large training set, due to its fast

Table 4. Storage requirements for the various methods as reported by [34] when all methods are trained on the PGS++ corpus.

PCFG	Markov	Hashcat	JtR	Neural	PESrank
4.7 GB	1.1 GB	756 MB	756 MB	60 MB	1.19 GB

Table 5. Overall performance comparison to existing methods.

	PCFG	Neural	Neural+MonteCarlo	PESrank
Training time	Hours/days	Hours/days	Hours/days	**12 min**
Lookup time	Offline	Offline	**Online**	**Online**
Tweak time	$\leq$ **1 s**	Hours/days	Hours/days	$\leq$ **1 s**
Storage requirement	Highest	Lowest	Lowest	Medium
Explainability	Maybe	No	No	**Yes**
Accuracy	**Exact**	**Exact**	Unknown	**up to 1 bit**

training time, whereas the other methods' ability to do so in reasonable time is currently unknown.

5.2 Storage Requirements

Table 4 summarizes the storage space of the different methods, as reported by [34]—where, unlike in the PGS service [5], the authors trained the earlier methods on the PGS++ training set. For comparison we provide the PESrank storage for the same set. The table shows that the Neural network requires the lowest amount of storage (60 MB) on the server-side, while PESrank requires a larger, yet very reasonable 1.19 GB storage, and significantly less than PCFG.

6 Related Work

Password strength measurement often takes one of two conceptual forms: heuristic pure-estimator approaches, and cracker-based approaches. Usually existing cracker-based methods are *generative*: they enumerate the passwords in their model either in their training phase or in their lookup phase. PESrank belongs to the cracker-based approaches, however, unlike previous methods, it is not generative. It's underlying rank estimation algorithm works directly on the multi-dimensional probability distribution *without enumerating*. This non-generative estimation is the reason why PESrank's training time and tweaking time are dramatically faster than those of [34,35,47]. In this section we describe earlier work on cracker-based approaches. Additional related work on pure-estimator methods, and about model tweaking, can be found in the Appendix.

Software tools are commonly used to generate password guesses [18]. The most popular tools transform a wordlist using mangling rules, or transformations intended to model common behaviors in how humans craft passwords.

Two popular tools of this type are Hashcat [21] and John the Ripper [36]. These tools typically run until a timeout is triggered. Since they generally take a long time to run (minutes to hours, depending on the timeout setting) their usefulness as online strength estimators is limited.

A probabilistic cracker method, based on a Markov model, was first proposed in 2005 [35], and studied more comprehensively subsequently [7,15,31]. Markov models predict the probability of the next character in a password based on the previous characters, or context characters. This method is generative: it calculates the rank of a given password by enumerating over all possible passwords in descending order of likelihood, which is computationally intensive, and makes it impractical as an online strength estimator. In addition, in order to tweak this method for each user (based the user's personal information), the model method should be retrained for each user separately [6]. Since the training takes days, it is unrealistic to tweak.

In 2009 Weir et al. [47] proposed a very influential method which uses probabilistic context-free grammars (PCFG). The intuition behind PCFG is that passwords are built with template structures (e.g., 6 letters followed by 2 digits) and terminals that fit into those structures. A password's probability is the probability of its template multiplied by those of its terminals. In 2015 the PCFG method was integrated with the techniques reported by Komanduri in his PhD thesis [28]. Conceptually, this method is similar to ours since it also assumes probability independence between model components: Our method assumes independence between the probabilities of its corresponding sub-passwords while PCFG assumes independence between the probability of the template and the terminals. Like the Markov method, the PCFG method is generative: it calculates the rank of a given password by enumerating over all possible passwords in descending order of likelihood, so it is also impractical as an online strength estimator. In contrast, PESrank calculates the rank of a given password in the descending order of likelihood *without enumerating* over the passwords themselves. Due to its 2-level model (template + terminals), which is fairly intuitive, we believe PCFG *may be* explainable—although its authors did not develop or discuss this capability.

In 2016 Melicher et al. [34] proposed to use a recurrent neural network for probabilistic password modeling. Like Markov models, neural networks are trained to generate the next character of a password given the preceding characters of a password. In its pure form this method is also generative and therefore is computationally intensive. However, the authors also describe a Monte-Carlo method to *estimate* the rank of a given password. To do so they split the algorithm into two phases: the training and sampling phase (which is generative), and a lookup phase, which uses the sampled model to provide an estimate. Therefore, like PESrank, in Monte-Carlo mode the Neural method's lookup is non-generative, making it suitable for online strength estimation. The authors do not provide bounds on the estimation error introduced by the Monte Carlo method. However, the Neural method's training phase *remains* generative: it enumerates the passwords to train the neural network. Thus, in order to personalize the neural network method for each user, it should be retrained for each user separately. Since the training takes days, this method cannot be personalized for different users in real-time. Moreover, like most neural-network-based systems, the algorithm is inherently difficult to explain, only providing a numeric rank without any hints about "why" or what to do to improve.

In Table 5 we summarize the overall differences between the leading methods according to several criteria: training time, lookup time, tweaking time, storage, explainabilty and accuracy. The information about PCFGs in this comparison is taken from [28,47] plus [23,29,45] regarding its tweakability. The table shows that the PESrank method outperforms all of the leading alternatives, in different ways. Versus PCFGs, PESrank is online and its training time is significantly shorter. Versus the Neural method in its pure variant again PESrank is superior since it is online, has shorter training time, plus it is tweakable and explainable. Finally, versus the Neural method's Monte-Carlo variant (which is practical as an online estimator), PESrank retains all its other advantages in training and tweak time, explainability, and provable accuracy.

7 Conclusions

In this paper we proposed a novel password strength estimator, called PESrank, which accurately models the behavior of a powerful password cracker. PESrank calculates the rank of a given password in an optimal descending sorted order of likelihood. It is practical for *online* use, and is able to estimate a given password's rank in fractions of a second. Its training time is drastically shorter than previous methods. Moreover, PESrank is efficiently *tweakable* to allow model personalization, without the need to retrain the model; and it is *explainable*: it is able to provide information on *why* the password has its calculated rank, and gives the user insight on how to pick a better password.

PESrank casts the question of password rank estimation in a multi-dimensional probabilistic framework used in side-channel cryptanalysis, and relies on the ESrank algorithm for side-channel rank estimation. We found that an effective choice uses $d = 5$ dimensions: base word, prefix, suffix, capitalization pattern, and l33t transformation. We implemented PERrank in Python and conducted an extensive evaluation study of it. We also integrated it into the registration page of a course at our university. Even with a model based on 905 million passwords, the response time was well under 1 s, with up to a 1-bit accuracy margin between the upper bound and the lower bound on the rank.

We conclude that PESrank is a practical strength estimator that can easily be deployed in any web site's online password registration page to assist users in picking better passwords. It provides accurate strength estimates, negligible response-time overhead, good explainability, and reasonable training time.

Acknowledgments. We thank Lujo Bauer and Michael Stroucken for allowing us broad use of the PGS service and assisting us in obtaining the PGS and PGS++ sets.

A Additional Related Work

A.1 Heuristic pure-estimator approaches

The earliest and probably the most popular methods of password strength estimation are based on LUDS: counts of lower- and uppercase letters, digits and symbols. The de-facto standard for this type of method is the NIST 800-63 standard [4,19]. It proposes to measure password strength in entropy bits, on the basis of some simple rules such as the length of the password and the type of characters used (e.g., lower-case, upper-case, or digits). These methods are known to be quite inaccurate [12].

Wheeler proposed an advanced password strength estimator [48], that extends the LUDS approach by including dictionaries, considering l33t speak transformations, keyboard walks, and more. Due to its easy-to-integrate design, it is deployed on many websites. The meter's accuracy was later backed up by scientific analysis [49].

Guo et al. [20] proposed a lightweight client-side meter. It is based on cosine-length and password-edit distance similarity. It transforms a password into a LUDS vector and compares it to a standardized strong-password vector using the aforementioned similarity measures.

Such pure-estimator approaches have the advantage of very fast estimation—typically in fractions of a second—which makes them suitable for online client-side implementation. However, they do not directly model adversarial guessing so their accuracy requires evaluation.

A.2 Tweakable extensions and variations

Several authors (cf. [23,29,45]) extended the PCFG approach to develop systems that also use personal information. The nature of the extensions was to add a new grammar variable for each type of personal information, (e.g., B for birthday, N for name and E for email) which makes the approach tweakable. However these extended methods are impractical for online use for the same reasons PCFG is impractical: they are all generative.

Personalized password strength meters (PPSMs) which rely on previous password knowledge have also been proposed [8,37]: PPSMs warns users when they pick passwords that are vulnerable based on previously compromised passwords. Similarly, PESrank can be personalized based on previous passwords, but also can be personalized based on any kind of user personal information (name, email, etc.).

Recently [22] introduced PassGAN, an approach that replaces human-generated password rules by machine learning algorithms. The PassGAN uses a Generative Adversarial Network (GAN) to learn the distribution of real passwords from actual password leaks, and to generate password guesses. The authors did not compare their results with previous rank estimators and did not report on the required training time.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Cryptographic Hardware and Embedded Systems-CHES 2002, 29–45 (2003)
2. David, L., Wool, A.: Online password guessability via multi-dimensional rank estimation. arXiv preprint arXiv:1912.02551 (2019)
3. Bogdanov, A., Kizhvatov, I., Manzoor, K., Tischhauser, E., Witteman, M.: Fast and memory-efficient key recovery in side-channel attacks. In: Selected Areas in Cryptography (SAC) (2015)
4. Burr, W., Dodson, D., Polk, W.: Electronic authentication guideline. Technical report, National Institute of Standards and Technology (2004)
5. Carnegie Mellon University Password Research Group. Password guessability service (pgs) (2019). https://pgs.ece.cmu.edu/
6. Castelluccia, C., Chaabane, A., Dürmuth, M., Perito, D.: When privacy meets security: Leveraging personal information for password cracking. arXiv preprint arXiv:1304.6584 (2013)
7. Castelluccia, C., Dürmuth, M., Perito, D.: Adaptive password-strength meters from markov models. In: NDSS (2012)
8. Das, A., Bonneau, J., Caesar, M., Borisov, N., Wang, X.: The tangled web of password reuse. In: NDSS 2014, pp. 23–26 (2014)
9. David, L., Wool, A.: A bounded-space near-optimal key enumeration algorithm for multi-subkey side-channel attacks. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 311–327. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_18
10. David, L., Wool, A.: Poly-logarithmic side channel rank estimation via exponential sampling. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 330–349. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_17
11. David, L., Wool, A.: PESrank Python implementation (2020). https://github.com/lirondavid/PESrank
12. From very weak to very strong: de Carné de Carnavalet, X., Mannan, M. Analyzing password-strength meters. In: NDSS **14**, 23–26 (2014)
13. Dell'Amico, M., Filippone, M.: Monte carlo strength evaluation: Fast and reliable password checking. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 158–169. ACM (2015)
14. Dell'Amico, M., Michiardi, P., Roudier, Y.: Password strength: an empirical analysis. In: 2010 Proceedings IEEE INFOCOM, pp. 1–9. IEEE (2010)
15. Dürmuth, M., Angelstorf, F., Castelluccia, C., Perito, D., Chaabane, A.: OMEN: Faster Password Guessing Using an Ordered Markov Enumerator. In: Piessens, F., Caballero, J., Bielova, N. (eds.) ESSoS 2015. LNCS, vol. 8978, pp. 119–132. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15618-7_10
16. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_21
17. Glowacz, C., Grosso, V., Poussier, R., Schueth, J., Standaert, F.-X.: Simpler and more efficient rank estimation for side-channel security assessment. In: Fast Software Encryption, pp. 117–129 (2015)
18. Goodin, D.: Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331". Ars Technica (2013)
19. Grassi, P.A., et al.: NIST special publication 800–63b: Digital identity guidelines (2017)

20. Guo, Y., Zhang, Z.: LPSE: lightweight password-strength estimation for password meters. Comput. Secur. **73**, 507–518 (2018)
21. hashcat. Hashcat advanced password recovery (2019)
22. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: PassGAN: a deep learning approach for password guessing. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 217–237. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_11
23. Houshmand, S., Aggarwal, S.: Using personal information for targeted attacks in grammar based probabilistic password cracking. In: IFIP Advances in Information and Communication Technology, vol. 511 (2017)
24. Jason. 1.4 billion leaked passwords in over 40GB of data (2019)
25. Kelley, P.G., et al.: Guess again (and again and again): measuring password strength by simulating password-cracking algorithms. In: 2012 IEEE Symposium on Security and Privacy, pp. 523–537. IEEE (2012)
26. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology-CRYPTO 1999, pp. 388–397. Springer (1999)
27. Kocher, P.C.: Timing attacks on implementations of diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
28. Komanduri, S.: Modeling the adversary to evaluate password strength with limited samples. Ph.D. thesis, Carnegie Mellon University (2016)
29. Li, Y., Wang, H., Sun, K.: Personal information in passwords and its security implications. IEEE Trans. Inf. Forensics Secur. **12**(10), 2320–2333 (2017)
30. Li, Z., Han, W., Xu, W.: A large-scale empirical analysis of Chinese web passwords. In: 23rd USENIX Security Symposium, pp. 559–574 (2014)
31. Ma, J., Yang, W., Luo, M., Li, N.: A study of probabilistic password models. In: 2014 IEEE Symposium on Security and Privacy, pp. 689–704. IEEE (2014)
32. Martin, D.P., Mather, L., Oswald, E.: Two sides of the same coin: counting and enumerating keys post side-channel attacks revisited. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 394–412. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_21
33. Martin, D.P., O'Connell, J.F., Oswald, E., Stam, M.: Counting keys in parallel after a side channel attack. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 313–337. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_13
34. Melicher, W., et al.: Fast, lean, and accurate: modeling password guessability using neural networks. In: Proceedings of 25th USENIX Security Symposium, pp. 175–191 (2016)
35. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, pages 364–372. ACM (2005)
36. OpenWall. John the ripper password cracker (2019)
37. Pal, B., Daniel, T., Chatterjee, R., Ristenpart, T.: Beyond credential stuffing: Password similarity models using neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 417–434. IEEE (2019)
38. Poussier, R., Standaert, F.-X., Grosso, V.: Simple key enumeration (and rank estimation) using histograms: an integrated approach. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 61–81. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_4

39. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45418-7_17
40. Shay, R., et al.: Encountering stronger password requirements: user attitudes and behaviors. In: Proceedings of the Sixth Symposium on Usable Privacy and Security (SOUPS 2010), p. 2. ACM (2010)
41. Ur, B., et al.: Design and evaluation of a data-driven password meter. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 3775–3786. ACM (2017)
42. Ur, B., et al.: How does your password measure up? the effect of strength meters on password creation. In 21st USENIX Security Symposium, pp. 65–80 (2012)
43. Veras, R., Collins, C., Thorpe, J.: On semantic patterns of passwords and their security impact. In NDSS (2014)
44. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.-X.: An Optimal Key Enumeration Algorithm and Its Application to Side-Channel Attacks. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 390–406. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_25
45. Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: an underestimated threat. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1242–1254 (2016)
46. Weir, M., Aggarwal, S., Collins, M., Stern, H.: 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, pp. 162–175. ACM (2010)
47. Weir, M., Aggarwal, S., De Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 391–405. IEEE (2009)
48. Wheeler, D.: zxcvbn: realistic password strength estimation. Dropbox TechBlog (2012)
49. Wheeler, D.L.: zxcvbn: low-budget password strength estimation. In: Proceedings of 25th USENIX Security Symposium, pp. 157–173 (2016)