**LPSE: Lightweight password-strength estimation for password meters**

Yimin Guo *, Zhenfeng Zhang

*Trust Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China

**ABSTRACT**

User-created strong passwords are the key to guaranteeing the security of password authentication. In practice, users often choose passwords that feel safe and that they can remember easily. However, the user’s perception of the strength of passwords is inconsistent with the actual strength of these passwords. To encourage users to create strong passwords, many websites use password meters to visualize the strengths of user-chosen passwords, whereas the existing password meters have limited accuracy. The state-of-the-art password-guessing approaches have high accuracy in testing the strengths of passwords, but these algorithms are not suitable for detecting user password strength directly on the client side, due to the long running time and the data storage problem. In this paper, we propose a lightweight password-strength estimation method (LPSE). By testing the strong and weak passwords selected by a state-of-the-art password cracking-algorithm, we observed that our LPSE algorithm is superior to the existing lightweight password-strength estimation algorithms in the accurate identification of strong passwords and weak passwords. Moreover, the LPSE algorithm requires notably little storage space and is sufficiently fast for client-side measurement of password strength.

© 2017 Elsevier Ltd. All rights reserved.

**Keywords:** Password meter, Password-strength estimation, Cosine-length similarity, Edit-length similarity, Password guessing

---

**1. Introduction**

Password authentication is the first line of defense in protecting network systems. Text passwords are a common form of authentication (Stobert and Biddle, 2014). To ensure the security of text-password authentication, many systems require users to choose strong passwords. User-created passwords are easy to remember, but are also vulnerable to guessing attacks (Das et al., 2014; Stobert and Biddle, 2014; Ur et al., 2016; Wash et al., 2016). To prevent the emergence of weak passwords, system administrators have adopted a variety of measures, including system-assigned passwords (Al-Ameen et al., 2015; Huh et al., 2015) and strict password composition policies (Inglesant and Sasse, 2010; Shay et al., 2016; Weir et al., 2010). System-assigned passwords can be hard to guess, but they are also difficult to remember for users. Password composition policies require that each password meet certain requirements (e.g. at least 8 characters in length and at least from three of four character classes), which can also make passwords difficult to guess (Komanduri et al., 2011). However, a strict password composition policy can lead to user frustration, and the user may choose the password in a simple and predictable way to meet the policy requirements (Komanduri et al., 2011; Shay et al., 2010). Another way to permit users to select strong passwords is to employ proactive password checking (Bishop and Klein, 1995). Recently, proactive password checkers are being deployed as password meters on many websites to encourage users to choose strong passwords (Shay et al., 2015). The password meter is usually a visual representation of password

---

* Corresponding author.

E-mail address: yiminguo_tca@163.com (Y. Guo).

https://doi.org/10.1016/j.cose.2017.07.012

0167-4048/© 2017 Elsevier Ltd. All rights reserved.
strength on the screen with text and color bars. Evidence shows that users are influenced by password meters in their password choices when informed about password strength (Egelman et al., 2013; Ur et al., 2012). Ur et al. (2012) found that various visual password meters encourage users to create longer passwords. The password meters also affect the users’ act of password creation, and lead users to choose different characters with which to build their passwords, such as digits, special symbols, and uppercase letters.

To help users choose stronger passwords, many websites have deployed password meters to provide visual feedback on password strength. However, previous studies have shown that the existing password meters have limited accuracy as they often label weak passwords as strong and mark strong passwords as weak (Castelluccia et al., 2012), and different password meters give highly inconsistent strength outcomes for the same password (de Carnavalet and Mannan, 2014, 2015; Ji et al., 2015). The limited accuracy of existing password meters may confuse users in choosing a stronger password. Most of the existing password meters are deployed on either the server or the client, and they employ rule-based methods to measure password strength. Simple rule-based strength-estimation algorithms cannot capture the complexity of passwords, making it difficult to accurately estimate the strengths of given passwords (Melicher et al., 2016). Recent researches (de Carnavalet and Mannan, 2015; Melicher et al., 2016) show that the zxcvbn algorithm (Wheeler, 2016) uses some reasonable evaluation rules, and gives more accurate strength estimates than other rule-based password-strength algorithms.

It is a challenging task to design a password-strength evaluation method that is suitable for password meters. Due to the complexity of passwords, there are still some deviations in strength measurements of the same password using the most advanced password-guessing techniques (Ur et al., 2015). We believe that an efficient lightweight password meter should be capable of accurately determining the strength of a proposed password on the user’s client machine. There are security and latency problems with the password meters deployed on the server side (Melicher et al., 2016; Van Acker et al., 2015). If the password-strength estimation is done on the client side, it is necessary to run the password strength meter in real time with storage space but also increases the searching time of the password-strength evaluation algorithms. In addition, a common password dictionary includes words in only one language, and the most common words in passwords in other languages are not recognized.

To evaluate accurately the strengths of user-created passwords in real time, we propose a lightweight password-strength estimation method (LPSE) that is suitable for running entirely on the client side. LPSE measures the strength of a password by comparing the similarity in the structure and the distance between a given password and a standard strong password. A large number of strong passwords and weak passwords, selected by the state-of-the-art password-cracking algorithm, were tested. We found that the similarity-evaluation method can more fully capture the strong and weak password features and can determine the password strength more accurately than the existing password-strength meters.

Our main contributions are the following.

1. We represent passwords by vectors that contain the components of the different character types and lengths that make up the password. By determining the similarity between a given password vector and the standard strong-password vector in terms of structure, vector modulus and password distance, it can accurately capture the characteristics of a given password from multiple perspectives.

2. The password-evaluation method proposed in this paper is lightweight: it not only can accurately identify the password strength but also has such advantages as small storage space, fast running speed, and ease of implementation. It is suitable for client-side password checking.

3. The accuracies of our scheme and several existing schemes are evaluated by performing a number of experiments. We compared the impacts of these schemes on the usability and security of user-chosen passwords, and found that LPSE has the lowest false-negative rate in determining the password strength.

2. Related work

There are two primary ways to measure password strength. The first approach relies on the complexity of the password itself, such as evaluating password strength as Shannon entropy (Bonneau et al., 2015; Lin, 1991) or using some statistical methods (Bonneau, 2012a, 2012b; Weir et al., 2010). The second approach is to simulate the adversary’s password-guessing capabilities, such as measuring password strength as the number of guesses that an attacker would need to guess a given password (Ur et al., 2015).

Early efforts to measure password strength often use password entropy (Florencio and Herley, 2007; Forget et al., 2008).
Many studies have shown that the password-entropy metric is not suitable for measuring the strengths of user-chosen passwords and is only suitable for assessing the strengths of randomly generated passwords (Egelman et al., 2013; Florêncio et al., 2014). For user-created passwords, the entropy metric needs to know the probability distribution of passwords. An accurate measure of the password probability distribution requires a large number of password samples (Bonneau, 2012a; Paninski, 2003). Password checkers for the password meters often use the “entropy” or “score” of the password to measure password strength, such as NIST password entropy (Burr et al., 2013) and Microsoft password checker. This type of approach is also referred to as rule-based password-strength measures. Rule-based methods measure the “entropy” of a password using various bonus and decrement rules. Password scores are usually determined by password length, the use of digits, lowercase letters, uppercase letters, and special characters in the password, and whether or not the password contains blacklisted words. zxcvbn (Wheeler, 2016) is a very good rule-based password measure algorithm, which divides a given password into several patterns, and then evaluates the entropy of each pattern separately. The following patterns are considered: repeat (e.g., 111, sss, 10gf10gf); sequence (e.g., 123, 246, efgh); reversed (to reverse a word, e.g., DrowssaP); keyboard (e.g., qwerty, qAxzcde3); date (e.g., 7/8/1990, 07081992). If any of these weak patterns appear in a given password, the algorithm only considers that particular “part” of the password to be drawn from a small space of possibilities in estimating the overall strength. zxcvbn also embeds multiple dictionaries to check the given password: if a subpart of the password is found in a dictionary, the entropy value is assigned based on the rank of the subpart in the dictionary. The remaining part of a given password that is not contained in the dictionary is considered a random string. The final password entropy is calculated as the sum of the entropy of each pattern.

Password guessing, which measures the strength of a password from the perspective of an attacker’s ability, is an approach that is in many ways similar to password estimating. The password-guessing techniques enumerate the guesses in descending order of probability, which means that most frequent passwords are first tested, and a password that is guessed later is stronger. Therefore, researchers often use the number of guesses to indicate password strength (Ur et al., 2015).

John the Ripper (JtR) (OpenWall, 1990) is one of the most popular password-cracking tools. It supports multiple modes to generate password guesses. In dictionary mode, a dictionary and various mangling rules are used to generate guesses. Mangling rules are used to model common behaviors in how users create passwords. They can generate variants of words from an input dictionary, which allows for more efficient guesswork. For example, a mangling rule may append a digit to a word, or replace the letter a with Ω.

At present, the state-of-the-art password-guessing algorithms are based on the probability password models (Narayanan and Shmatikov, 2005; Weir et al., 2009). Probabilistic approaches establish a model by using plaintext password training data (usually obtained from publicly leaked passwords), in which each password has a given probability of being selected, and then enumerate the guesses in descending order of probability. Probabilistic attack methods can greatly reduce the number of guesses for long passwords and/or passwords that do not appear in dictionaries. The most well-known password probabilistic attack methods are probabilistic context-free grammars (Weir et al., 2009) and methods based on Markov models (Narayanan and Shmatikov, 2005). The underlying idea of probabilistic context-free grammar is that a typical password has a certain structure, and the probabilities of different structures are extracted from the list of leaked passwords, which are then used to generate password guesses. The technique consists of two steps: the first is constructing the probabilistic context-free grammar from a leaked password training set, and the second is automatically extracting mangling rules by training the context-free grammar, and generating the actual guesses in descending order of probability. Narayanan and Shmatikov’s algorithm cannot enumerate passwords in descending order of probability. Dürmuth et al. (2015) then implemented this feature.

To compute directly the number of guesses for a given password, Kelley et al. (2012) designed a guess-number calculator for password-guessing algorithms. It maps a password to the number of guesses required to guess the password. The guess-number calculator does not need to run a guessing algorithm but can figure out the number of guesses for each password. For example, for the PCFG algorithm, the guess-number calculator creates a lookup table based on a training set, and then computes the number of guesses for each password. However, it is computationally slow to create this lookup table. Dell’Amico et al. (2010) adopted an approximation technique to estimate the number of guesses when using n-gram models. Dell’Amico and Filippone (2015) proposed a method to count the number of guesses for a given password that does not need to run a password-guessing algorithm. This method requires less training-data resources, and has good convergence properties. Castelluccia et al. (2012) presented the design of password meters using Markov models. This approach adopts the Markov model to assign probabilities for each password, and then takes the probability of the password as the password strength. Melicher et al. (2016) proposed a password-guessing attack method using an artificial neural network, and the neural network can be compressed to several hundred kilobytes without substantially diminishing the effectiveness of password guessing. Similar to Markov models, neural networks in this method are trained to generate the next character of a password given the preceding characters of a password.

Rule-based algorithms estimate strength from the perspective of the password structure and some weak password patterns, and their accuracies depend on the ability to capture the characteristics of the user-chosen passwords. Due to the complexity of the passwords that the users choose, the current rule-based password-strength measure is not sufficiently accurate (de Carnavalet and Mannan, 2015). Password-guessing techniques are considered the most accurate measures of password strength (Dell’Amico and Filippone, 2015; Kelley et al., 2012), but these techniques require very expensive computational efforts and very large disk space, thus they
are not well-suited to measure password strength for password meters.

To design an effective password-strength evaluation method that can run on the client side, we propose a similarity-evaluation method, which measures password strength by evaluating the similarity between a given password and a standard strong password in many ways. The similarity method evaluates password strength from the aspects of password structure, password vector modulus and password distance; therefore, it can capture the characteristics of a password more comprehensively than the existing rule-based password-strength methods, and it has higher accuracy. From the experimental results, it can be seen that the false-negative rate of the similarity-evaluation method is significantly lower than those of other methods.

3. Our LPSE algorithm

3.1. Overview

Our LPSE algorithm adopts two kinds of similarity to measure the strength of a given password, namely, cosine-length similarity and password-edit distance similarity. The password is represented by a vector that consists of digits, lowercase letters, uppercase letters, special characters, and the length of the password. Password strength can be measured by comparing the similarity between a given password vector and a standard strong-password vector.

We determine the similarity between the two password vectors from three aspects: the structure of the password (that is, what kinds of characters compose the password and the proportions of various characters); the password length; and the number of insertion, substitution, and deletion operations required to transform a given password into a standard strong password. Obviously, if a given password is more similar to a standard strong password in terms of structure and password length, and the number of insertion, substitution, and deletion operations required to transform it into a standard strong password is lower, then the password is stronger.

Cosine similarity determines the difference in direction between the two vectors, so it can be used to check whether the two password vectors are similar in structure (Bayardo et al., 2007). However, cosine similarity cannot determine the similarity in length between the two vectors. To this end, we propose a cosine-length similarity, which is an improved method of cosine similarity. The cosine-length similarity can determine the similarity between two password vectors in password structure and password-vector modulus.

The edit distance is the minimum number of operations required to convert a string into another string by inserting, replacing and deleting (Ristad and Yianilos, 1998). Since the standard strong password refers to a class of passwords with certain characteristics, rather than a specific password, the improved password-edit distance is the minimum number of insertion, substitution, and deletion operations required to transform a given password into a standard strong password. The smaller the edit distance between the given password and a standard strong password the greater the similarity between the two passwords and, hence, the given password may be assumed to be a strong password.

For a given password $\alpha$, its cosine-length similarity to the standard strong password is expressed as $s_{cl}(\alpha)$, and its password-distance similarity to the standard strong password is denoted as $s_{p}(\alpha)$. Then the strength of the password $\alpha$ is represented as $T_\alpha = (s_{cl}(\alpha), s_p(\alpha))$.

3.2. Password-vector calculation

The numbers of different types of characters appearing in the password are mapped to a frequency, together with the password length. A password can be expressed as a vector $\alpha = (x_1, x_2, x_3, x_4, x_5)$, where $x_1, x_2, x_3, x_4, x_5$ represent, respectively, the vector components of digits, lowercase letters, uppercase letters, special characters, and password length. The general rules for mapping different types of characters to vector values are shown in Table 1. The vector values for uppercase letters and special characters are larger than the vector values for digits and lowercase letters, since passwords that contain uppercase letters and special characters are stronger (Li et al., 2014; Ur et al., 2015). The vector value for two identical or consecutive characters is equivalent to the vector values of a character, such as $‘1’$ and $‘1’$ are the same vector values, $‘A’ B$ and $‘A’$ both have the same vector values. Assuming a password is $‘a’ ‘a’ ‘s’ ‘t’ ‘x’ ‘1’$, the vector values of this password string can be expressed as $\alpha = (3, 1, 4, 3, 8)$.

Many studies have shown that users prefer to choose some weak password patterns (Das et al., 2014; Ji et al., 2015; Li et al., 2014). In the calculation of a given password vector, we first determine whether there is a weak password pattern, and the weak pattern will be assigned a low vector value. The common weak password patterns are shown in Table 2.

Certain password-strength evaluation algorithms often use a dictionary to check whether a given password contains common words, such as $‘x’ ‘c’ ‘v’ ‘b’ ‘n’$ (Wheeler, 2016). The advantage of using a dictionary check is that when the user chooses a password containing common words, they can be quickly identified, but the use of the dictionary will take up a certain amount of storage space, and there are such issues as slow matching and low coverage. In addition, when the user chooses a word that is not in the checked dictionary, the system determines that it is a random word.

To identify whether a password contains common words, we determine whether a password contains the most common

<table>
<thead>
<tr>
<th>Patterns</th>
<th>Vector value</th>
<th>Example (character → vector value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>1</td>
<td>8 → 1</td>
</tr>
<tr>
<td>Lowercase letters</td>
<td>1</td>
<td>d → 1</td>
</tr>
<tr>
<td>Uppercase letters</td>
<td>2</td>
<td>G → 2</td>
</tr>
<tr>
<td>Special characters</td>
<td>3</td>
<td>&amp; → 3</td>
</tr>
<tr>
<td>Two identical characters</td>
<td>Equivalent to one character vector</td>
<td>aa → 1, 3a → 2</td>
</tr>
<tr>
<td>Two consecutive characters</td>
<td>Equivalent to one character vector</td>
<td>AB → 2, 1a → 2</td>
</tr>
</tbody>
</table>
two-character combinations, three-letter combinations, and starting and ending letters. The 30 most frequently occurring two-letter combinations in English include: the ing and her ere ent th the nth was eth for dth hat she ion int his sth ers ver. Half of all English words begin with t, a, s, or u, and half end in e, s, d, or t. If these frequently occurring patterns are found, they are mapped to the lowest possible values. The password-vector mapping rules for the weak mode are shown in Table 3.

### 3.3. Standard strong-password vector

The probability distribution of the most secure passwords is evenly distributed, so we select the standard strong password that satisfies two conditions:

1. The password is randomly generated.
2. The password is sufficiently long (e.g. at least 12 characters in length).

We believe that if a password is sufficiently long and the different types of characters that make up the password string and of equal probability then we consider the password to be strong. For example, for a randomly generated password with a length of L, and drawn from an alphabet of C characters, each password is of equal probability (C^-L). An attacker cannot search the password space in descending order of probability, and would have no better strategy than to guess passwords at random, so the search cost is C^L. The length of a strong password should also meet certain requirements. In their study, Shay et al. (2014) found that simple 16-character passwords are stronger than short complex passwords. Most passwords are less than 12 characters long based on the passwords that have been leaked, and only a small number of passwords are longer than 16 characters (Ji et al., 2015). Based on this evidence, we believe that a strong password should be randomly generated, and the password length should be greater than 16 characters. For example, to construct a standard strong password with a length of 18 characters, suppose there are 94 characters that can be used to form the password and these 94 characters consist of 10 digits, 26 lowercase letters, 26 uppercase letters, and 32 special characters. Since each character appears with equal probability, a strong password with a length of 18 characters should contain 2 digits, 5 lowercase letters, 5 uppercase letters, and 6 special characters. The password vector is formed according to the frequencies of the characters and their weights. Thus a standard strong password vector with a length of 18 characters can be $\alpha = (2,5,10,18,18)$, where the component values respectively represent digits, lowercase letters, uppercase letters, special characters, and password length.

#### 3.4. Password similarity

##### 3.4.1. Cosine-length similarity

Assuming that a given password vector is denoted by $\alpha = (x_0, x_1, x_2, x_3, x_4)$, and the standard strong-password vector is denoted by $\alpha_s = (y_0, y_1, y_2, y_3, y_4)$, the cosine similarity of the two vectors is:

$$\cos(\theta) = \frac{\sum_{i=0}^{5} (x_i \times y_i)}{\sqrt{\sum_{i=0}^{5} (x_i)^2} \times \sqrt{\sum_{i=0}^{5} (y_i)^2}}$$

The cosine similarity can only represent the structural similarity between two passwords, and does not reflect the similarity in vector modulus between the two passwords. Considering the

---

**Table 2 – The common weak password patterns.**

<table>
<thead>
<tr>
<th>Patterns</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password string contains common words</td>
<td>Password “Password123” contains the “password”</td>
</tr>
<tr>
<td>Password string matches the user name</td>
<td>Assuming the user name is zhoun and the password is zhoun123</td>
</tr>
<tr>
<td>Password string matches the registered site name</td>
<td>Assuming the registration is in hotmail and the password is hotmail123</td>
</tr>
<tr>
<td>Date pattern</td>
<td>910212, 20120828, 070867</td>
</tr>
<tr>
<td>Leeting pattern</td>
<td>a→@, s→$, 1→!</td>
</tr>
</tbody>
</table>

**Table 3 – Vector-mapping rules for weak-password patterns.**

<table>
<thead>
<tr>
<th>Weak-password patterns</th>
<th>Vector calculation</th>
<th>Example (password: character→vector value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The most common 2-letter and 3-letter combinations, and starting and ending letters</td>
<td>2-letter and 3-letter combinations are each calculated as one letter vector; Starting and ending letters are mapped to 0 Equivalent to a continuous character vector (see Table 1)</td>
<td>Goodidea47: ea→1</td>
</tr>
<tr>
<td>Keyboard pattern</td>
<td></td>
<td>estimation33: ion→1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>password123: ds→0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>quer123: qu→1,er→1</td>
</tr>
<tr>
<td>Password string matches the user name</td>
<td>User name is equivalent to a lowercase-letter vector Site name is equivalent to a lowercase-letter vector</td>
<td>zhoun356: zhoun→1</td>
</tr>
<tr>
<td></td>
<td>Equivalent to a 3-digit vector Only the 3 most common leeting transforms (a→@, s→$, 1→!) are considered. Equivalent to a lowercase-letter vector</td>
<td>(zhoun356 is a password, zhoun is the user name)</td>
</tr>
<tr>
<td>Date pattern</td>
<td></td>
<td>hotmail123: hotmail→1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(hotmail123 is a password, hotmail is the site name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>07081967: 07081967→3</td>
</tr>
<tr>
<td>Leeting pattern</td>
<td></td>
<td>P@ssword123: @→1</td>
</tr>
</tbody>
</table>
following special calculation rules.

The improved password-distance similarity is the minimum cost for transforming a given password into a standard strong password by insertion, deletion, and substitution operations. We specify that any two adjacent characters in a standard strong password are of different types.

The edit-distance operation rules for converting a given password to a standard strong-password type are as follows:

1. The edit-distance rule for a deletion is a subtraction operation: for the deletion of a digit or lowercase letter, subtract 1 from the edit distance; for the deletion of an uppercase letter, subtract 2 from the edit distance; for the deletion of a special character, subtract 3 from the edit distance.

2. The edit-distance rule for an insertion is an addition operation: for the insertion of a digit or a lowercase letter, add 1 to the edit distance; for the insertion of an uppercase letter, add 2 to the edit distance; for the insertion of a special character, add 3 to the edit distance.

3. The edit-distance rule for a substitution may be an addition or a subtraction operation. If a lowercase letter or a digit is replaced with an uppercase letter, the edit distance is incremented by 1. If a lowercase letter or a digit is replaced with a special character, the edit distance is incremented by 2. Lowercase letters are converted to digits, and digits are converted to lowercase letters, without changing the edit distance. If a special character is replaced with a lowercase letter, the edit distance should be decremented by 2.

The password-edit distance also takes into account the following special calculation rules.

1. If adjacent characters in a password are of different types, the edit distance is reduced by 0.5 for each occurrence; if adjacent characters are of the same type, the edit distance is increased by 0.5 for each occurrence.
2. If the given password length is less than 7 characters, the edit distance is set to the maximum value.
3. If the given password contains only one type of character, the edit distance is increased by 6; if it includes two types of characters, the edit distance does not change; if it includes three types of characters, the edit distance is decreased by 2; and if it includes 4 types of characters, the edit distance is decreased by 6.

The greater the edit distance between a given password and a standard strong password, the greater the difference between the two passwords. Conversely, smaller edit distance between a given password and a standard strong password denotes that the given password contains the properties of a standard strong password. Therefore, the similarity between them should be smaller. Using $\text{passworddist}(P, T)$ to denote the edit distance between a given password $P$ and a standard strong password $T$, the password-edit distance similarity is:

$$\text{Passwordsimilarity} = 1 - \frac{\text{passworddist}(P, T)}{\max(|P|, |T|)}.$$
chosen from the 7k7k and Yahoo leaked password lists as test dataset 1, and 999,211 passwords as test dataset 2 were selected from the leaked password lists of Duduniu, 178, abc, aha, casio, helfire, mayhem, opnkorea, rootkit, sunrise, tomsawyer, walla and whitefox. The details of the training dataset and test dataset 1 and 2 are summarized in Table 4.

<table>
<thead>
<tr>
<th>Data</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training dataset</td>
<td>RockYou, Tianya</td>
</tr>
<tr>
<td>Test dataset 1</td>
<td>7k7k, Yahoo</td>
</tr>
<tr>
<td>Test dataset 2</td>
<td>Duduniu, 178, abc, aha, casio, helfire, mayhem, opnkorea, rootkit, sunrise, tomsawyer, walla and whitefox</td>
</tr>
</tbody>
</table>

4.2. Testing process

We calculate the number of guesses of passwords using the Monte Carlo method (Dell’Amico and Filippone, 2015), which does not need to run a probability-guessing algorithm to calculate the number of guesses for a given password, and only a small amount of training data is required. The testing process consists of three steps. First, the PCFG algorithm is trained on the training data set, and the Monte Carlo method is used to calculate the numbers of guesses of test passwords. Second, according to the numbers of guesses of passwords, the strong and weak passwords are selected from test dataset 1 and the threshold values of the LPSE algorithm are determined by using these strong and weak passwords. Finally, the strong and weak passwords are selected from test dataset 2, and the performances of the LPSE algorithm and other password-strength evaluation algorithms are compared using these strong and weak passwords.

The number of guesses for a password is often used to determine the strength of the password (Ur et al., 2015). Florêncio et al. (2014, 2016) believe that to resist online attacks a password must survive $10^6$ guesses and to withstand offline attacks a password must survive $10^4$ or more guesses. Since the number of guesses for a password is related to the password-guessing algorithms and the training dataset, we choose the weak and strong passwords according to the following conditions. We propose a weak password cannot resist more than $10^{10}$ guesses, and a strong password should meet three conditions: (1) it must be able to survive at least $10^{14}$ guesses; (2) it must contain at least two different types of characters, or its length must be at least 12; (3) it must contain at least five different characters (for example, a and b are different characters).

4.3. Determining the LPSE thresholds

The standard strong-password vector $\alpha_0 = (2, 5, 10, 18, 18)$ is taken as an example to confirm the threshold values of the LPSE algorithm. There are a total of 198,270 passwords in test dataset 1; 152,530 passwords remain after deleting repeated passwords and unrecognized garbled characters. The method described in Section 4.2 is used to determine that test dataset 1 contains 5960 strong passwords and 120,419 weak passwords. The two similarities of the LPSE algorithm are tested with these selected strong and weak passwords, and the test results are shown in Fig. 1. It can be seen from Fig. 1(a) that the cosine-length similarities of the strong passwords are mostly within the interval [0.2–0.6], and the cosine-length similarities of the weak passwords are within the interval [0.13–0.3]. If the cosine-length similarity is greater than 0.3, or less than 0.2, we can distinguish whether a given password is strong or weak. We can also see some overlap between the values 0.2 and 0.3 of cosine-length similarity. In other words, the cosine-length similarity within this range is unable to solely distinguish strong passwords and weak passwords. From Fig. 1(b), the password-distance similarities of strong passwords are mainly in the range of [0.45–0.8], while those of the weak passwords are mainly distributed between [0.05–0.46]. It can be seen that a reasonable choice of the value of the password-distance similarity can distinguish strong passwords and weak passwords.

There are some differences between the two similarities of strong and weak passwords. The distance similarity and cosine-length similarity of a strong password are basically proportional to each other (Fig. 1(c)), but the two similarities of a weak password do not have a directly proportional relation (Fig. 1(d)). For some weak passwords with large cosine-length similarity, the password-distance similarity is relatively small, and the password-distance similarity is large for some weak passwords whose cosine-length similarity is small. There are a few outliers for which the two similarities are relatively large for weak passwords. The different characteristics of the two similarities for strong and weak passwords make it easy to determine the thresholds to distinguish strong passwords from weak passwords.

Based on the above test results, we give a function to distinguish strong from weak passwords as follows:

$$T_{\alpha} = (s_{\alpha}(\alpha), s_{\beta}(\alpha)) = \begin{cases} \text{strong} & \text{if } s_{\alpha}(\alpha) \geq 0.4 \text{ or } s_{\beta}(\alpha) \geq 0.55 \\ \text{strong} & \text{if } 0.3 \leq s_{\alpha}(\alpha) < 0.4 \\ \text{weak} & \text{and } 0.4 \leq s_{\beta}(\alpha) < 0.55 \\ \text{medium} & \text{otherwise} \end{cases}$$

where $s_{\alpha}(\alpha)$ and $s_{\beta}(\alpha)$ represent the cosine-length similarity and password-distance similarity of the password, respectively.

4.4. Performance-comparison results

We compare the performance of the LPSE algorithm with those of PM (Passwordmeter) and zxcvbn (Wheeler, 2016). The password-strength scale of the LPSE algorithm is Weak, Medium, Strong.

The strength scales of the PM algorithm and zxcvbn algorithm are expressed in terms of 5-item scales; for comparison, the password strength is handled in accordance with the 3-item scale for all algorithms. The “Very Weak” and “Weak” scale categories of the PM algorithm are denoted as “Weak”, “Good” as “Medium”, and “Strong” and “Very Strong” as “Strong”. Likewise, the “Very Weak” and “Weak” scale categories of the zxcvbn algorithm are denoted as “Weak”, “So-so” is expressed as “Medium”, and “Good” and “Great” are denoted as “Strong”. We use the 999,211 passwords in test dataset 2 to analyze the effects of the password-strength algorithms. After deleting repeated
passwords and unrecognized garbled characters, the number of valid passwords in test dataset 2 is 651,854. In accordance with the criteria for determining strong and weak passwords given in Section 4.2, test dataset 2 has 48,813 strong passwords, 460,156 weak passwords, and 142,885 medium passwords. The password-strength distribution in test dataset 2 is shown in Fig. 2.

The selected strong and weak passwords are used to test the accuracies with which LPSE, PM and zxcvbn evaluate these passwords. The accuracy of a good password-strength evaluation approach should be close to those of the most advanced password-guessing techniques. The results of identifying the strong and weak passwords with these three algorithms are shown in Figs. 3 and 4, which clearly indicate the advantages of LPSE for identifying strong and weak passwords.

The results of the performance comparison of the three algorithms are shown in Table 5, where $\beta_{SW}$ is the false-negative rate for strong passwords that the password-evaluation algorithm identified as weak passwords; $\beta_{WS}$ is the false-negative rate for weak passwords that the password-evaluation algorithm identified as strong passwords; and $\beta_{WS-M}$ is the rate at which weak passwords were identified by the password-evaluation algorithm as strong passwords or medium passwords. The running time is the time required by the password-strength algorithm to determine the strength of a password, which is the average time required to run 10 consecutive tests of 290,786 passwords in the configuration of CPU AMD A4-4300M 2.50G, RAM 4G.

As seen from Table 5, $\beta_{SW}$ and $\beta_{WS}$ of the LPSE algorithm are significantly smaller than those of the other two algorithms, that is to say, the LPSE algorithm is closest to the state-of-the-art password guessing techniques in accurately determining the password strength. An interesting phenomenon is that the zxcvbn algorithm has the highest $\beta_{WS}$ value, which means that the algorithm has the highest proportion of strong passwords misjudged as weak passwords. When strong passwords contain several weak patterns, the zxcvbn algorithm gives very low “entropy”, which may be due to the zxcvbn algorithm being too severe for weak password patterns.

It can also be seen from Table 5 that the effects of the three algorithms on the usability and security of user-chosen passwords are different. Websites or organizations generally reject weak passwords and accept medium or strong passwords. If.
a password-strength evaluation method gauges the strong password selected by the user as a weak password, the strong password is rejected, which reduces the usability of passwords. If a password-strength evaluation method identifies a weak password as a strong password or a medium password, this weak password is accepted, which decreases the security of passwords.

The value of $\beta_{SW}$ can reflect the impact of a password-evaluation algorithm on the usability of passwords. $\beta_{SW}$ is smallest for LPSE among the three algorithms, indicating that the LPSE algorithm has the lowest false-positive rate for strong passwords. $\beta_{WSM}$ can reflect the impact of a password-evaluation algorithm on the security of passwords. LPSE has the smallest $\beta_{WSM}$, indicating that it has the lowest misjudgment rate for weak passwords. As seen from Table 5, the storage space required by the LPSE algorithm is 33 k, and the running time for measuring the strength of a password is 0.181 ms, which is suitable for evaluating the password strength on the client side.

Here we compare the relationship between LPSE, PM, zxcvbn and the ideal password strength function using Spearman’s rank correlation. The Spearman’s rank correlation coefficient is a nonparametric technique for evaluating the degree of linear association or correlation between two independent variables (Gautheir, 2001). The basic idea is that for a given password test set, the password-strength measure algorithm can sort the passwords’ strength values in the password set. The greater the Spearman’s correlation coefficient between the two strong sequences output by a password-strength measure method and the ideal password measure method, the closer the accuracy of the two measure methods is. Spearman’s correlation coefficients are between $-1$ and $1$, where $1$ means that the two sequences are completely positive correlation, $-1$ indicates perfect negative correlation, and 0 indicates no correlation.

The optimal strategy for an attacker is to guess passwords in descending order of probability. Guessing entropy represents

<table>
<thead>
<tr>
<th>Table 5 – Performance comparison of the three algorithms.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>LPSE</td>
</tr>
<tr>
<td>PM</td>
</tr>
<tr>
<td>zxcvbn</td>
</tr>
</tbody>
</table>
the average number of guesses an attacker needs to guess passwords in an optimal way. Let $\chi$ be a probability distribution, the sample space size is $N$, and the event probability $p_i$ in the probability distribution $\chi$ satisfies the monotonically decreasing sequence $p_1 \geq p_2 \geq \ldots \geq p_N$. The guessing entropy $G(\chi)$ is defined as:

$$G(\chi) = \sum_{i=1}^{N} i \cdot p_i.$$ 

If the order of the output of a function is the same as that of the optimal password guessing sequence, the function can be used as an ideal password strength measure function. The optimal password guessing order is guessed in descending order of probability, so we can use the probability of each password in the password set to design an ideal password strength function. Assuming that each password has a fixed probability on a given space of passwords, an ideal password strength function is (Castelluccia et al., 2012):

$$f(x) = -\log(p(x)).$$

The PCFG algorithm is used to determine the probability of passwords in the 651,854 test dataset 2, which can be approximated as an ideal password strength measure function. In the PCFG algorithm, passwords are grouped by templates, and the guessing probability of a given password is the frequency of the password templates multiplied by the frequency of each pattern in the template, such as $P(\text{guo@126}) = P(L_3 S_1 D_3) \times P(L_3 \rightarrow \text{guo}) \times P(S_1 \rightarrow @) \times P(D_3 \rightarrow 126)$, where $P(L_3 \rightarrow \text{guo})$ is the frequency of “guo” in three-lowercases groups, $P(S_1 \rightarrow @)$ is the frequency of “@” in one-special character groups, and $P(D_3 \rightarrow 126)$ denotes the frequency of “126” in three-digits groups.

Fig. 5 shows the Spearman’s correlation of the password cosine-length similarity, the password-distance similarity, the zxcvbn and PM, and the ideal password strength function. As a result of the test, it can be seen that the accuracy of cosine-length similarity is superior to that of other methods. The distance similarity, zxcvbn and PM perform similarly poor performance. LPSE algorithm is a combination of the password cosine-length similarity and password-distance similarity, reasonably set the thresholds of two similarities, and can therefore more accurately determine whether a password is strong password or a weak password. But LPSE algorithm sometimes cannot distinguish which password is stronger or weaker. For example, suppose the strength of the password $\alpha_1$ is (0.51, 0.43), the strength of the password $\alpha_2$ is (0.50, 0.45). According to the strong and weak passwords threshold given in Section 4.3, we can know that $\alpha_1$ and $\alpha_2$ are all the strong passwords, but we are unable to determine which one is stronger. This deficiency of the LPSE algorithm does not prevent it to become an excellent password checker for password meter. We believe that a general password meter only needs to determine whether a proposed password is strong or weak and does not need to determine which password is stronger or weaker.

5. Conclusions

We propose a new lightweight password-strength measurement approach that can comprehensively capture the complexity of user-chosen passwords by comparing the similarity of a given password with a standard strong password. Our LPSE meter’s accuracy is significantly higher than those of the existing password-evaluation methods for password meters. Using the most advanced password-guessing algorithm to select the strong and weak passwords, the false-negative rates for the strong passwords and weak passwords of LPSE are obviously less than those of other algorithms. The LPSE algorithm has a memory size of only 33 k and an average run time of 0.181 ms, which is suitable for assessing the strength of user-chosen passwords on the client side.

REFERENCES

Al-Ameen MN, Fatema K, Wright M, Scielzo S. The Impact of Cues and User Interaction on the Memorability of


Yimin Guo was born in 1992. She received the M.S. degree in department of computer science from Shaanxi Normal University. She is currently pursuing the Ph.D. degree at Trusted Computing and Information Assurance Laboratory, Institute of Software from Chinese Academy of Sciences. Her research interests include passwords, modern cryptography and information security.

Zhenfeng Zhang received the Ph.D. degree in Chinese Academy of Sciences. He is now a Research Professor at Trusted Computing and Information Assurance Laboratory, Institute of Software from Chinese Academy of Sciences. His research interests include theoretical cryptography, applied cryptography, security model and provable security of security protocols.