RLS-PSM: A Robust and Accurate Password Strength Meter Based on Reuse, Leet and Separation

Qiying Dong, Chunfu Jia, Fei Duan, and Ding Wang

***Abstract*— Password strength meters (PSMs) are being widely used, but they often give conflicting, inaccurate and misleading feedback, which defeats their purpose. Except for fuzzyPSM, all PSMs assume passwords are newly constructed, which is not true in reality. FuzzyPSM considers password reuse, six major leet transformations and initial capitalization, and performs the best as evaluated by Golla and Dürmuth at ACM CCS'18. On the basis of fuzzyPSM, we propose a new PSM based on Reuse, Leet and Separation, namely RLS-PSM. First, we classify password reuse behaviors into capitalization and those that use special characters for leet or separation, and calculate the corresponding probabilities. Then, to balance efficiency and precision, we use Long Short-Term Memory to calculate the probabilities of alphanumeric strings. Besides, we propose to use *benchmark passwords* to show the *relative strength* of a password. Due to the varied impacts of different service types and diversified economic value of websites, we consider parameter settings of RLS-PSM under six different service types. Finally, we use the Monte Carlo method and weighted Spearman coefficient to measure and compare the robustness and accuracy of RLS-PSM, leading PSMs (including Markov-based PSM, PCFG-based PSM, fuzzyPSM, RNN, and Zxcvbn), and password cracking tools (including JtR and Hashcat). We find that the robustness of RLS-PSM is significantly higher than all counterparts when *evaluating attempts* $> 10^4$ (e.g., on average, Fraction of Successfully Evaluated passwords of RLS-PSM is 18.9% higher than fuzzyPSM). The accuracy of RLS-PSM is also better than other mainstream PSMs used for comparison in this paper, except for fuzzyPSM.**

***Index Terms*— Password strength meters, password reuse, leet transformations, separation, relative strength.**

I. Introduction and Related Work

PASSWORDS are the most widely used method of identity authentication [1], which are the primary defense to maintain the security of systems and protect users' privacy [2]. With the rapid development of Internet technologies, more and more services require users to log in with their passwords. However, many users tend to create weak passwords that are easy to guess, or use the same password repeatedly on multiple websites [3], [4]. These vulnerable behaviors bring serious security risks to both systems and users.

To address this issue, mainstream Internet service providers use password strength meters (PSMs) to provide feedback of password strength to users when users register or modify their passwords. Only the PSM with accurate strength feedback can significantly improve the strength of users' passwords [5]–[7]. However, the widely used PSMs often provide inaccurate and misleading feedback, and the evaluation results of different websites also contradict each other [20]. Thus, when faced with real-world guessing attacks, the claimed security given by these PSMs may be overestimated. Table I shows the inconsistencies of strength evaluation for the same password from different high-profile websites.

For example, for a common password `Password`, Taobao evaluates it as "reject" and "few character types", Amazon evaluates it as "accept", and 126 evaluates it as "accept" and "medium security". This proves that even for websites of the same service type (e.g., Taobao and Amazon), the results of strength evaluation of the same password vary greatly. Besides, users often mistakenly believe that only appending digits or special characters at the end of a password, or simply capitalizing the initial, can upgrade the original weak password to a strong password [8], [9]. For instance, Taobao evaluates

TABLE I

Strength Feedback of Different Service Types of High-Profile Websites to Common Passwords†

Service type	Portal	Commercial		Email		Game		Forum	Social	
Website	Yahoo!	Tao.	Amz‡	126	Gmail	7k7k	178	CSDN	TY	RR
12345678	× TS	× F	✓	✓ W F	× W	✓ TO	× SEQ	× TS	× SEQ	✓ W
5201314	× TS	× F	✓	✓ W F	× TS	✓ TO	✓	× TS	× TO	✓ W
woaini	× TS	× F	✓	✓ W F	× TS	✓ TO	✓	× TS	× F	✓ W
iloveyou	× TS	× F	✓	✓ W F	× W	✓ TO	✓	× TS	× F	✓ W
password	× TS	× F	✓	✓ W	× W	✓ TO	✓	× TS	× F	✓ W
Password	× TS	× F	✓	✓ M	× W	✓ TO	✓	× TS	× F	✓ M
password123	× TO	✓ M	✓	✓ M	× W	✓ M	✓	✓ W	✓	✓ M
P@ssword	✓	✓ M	✓	✓ S	× W	× NA	✓	× TS	✓	✓ S
P@ssword123	✓	✓ S	✓	✓ S	× W	× NA	✓	✓ S	✓	✓ S
wang@123	✓	✓ M	✓	✓ S	✓ FA	× NA	✓	× TS	✓	✓ S
Wang@123	✓	✓ S	✓	✓ S	✓ G	× NA	✓	× TS	✓	✓ S

† Tao.: Taobao, Ama: Amazon, TY: Tianya, RR: Renren. ✓:Accept, ×: Reject, TS: Too short, W: Weak, M: Medium, S: Strong, SEQ: Continuous sequences of letters or digits, F: Few character types, NA: Some characters are not allowed, TO: Too obvious, FA: Fair, G: Good.

‡ Apart from Amazon, all the other websites in this table have leaked passwords which will be used in the later part of this paper.

Manuscript received January 14, 2021; revised May 11, 2021 and July 9, 2021; accepted August 5, 2021. Date of publication August 24, 2021; date of current version October 28, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFA0704703 and in part by the National Natural Science Foundation of China under Grant 62172240. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Issa Traore. *(Corresponding authors: Chunfu Jia; Ding Wang.)*

Qiying Dong and Ding Wang are with the College of Cyber Science, Nankai University, Tianjin 300350, China, also with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China, and also with the Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China (e-mail: dqy@mail.nankai.edu.cn; wangding@nankai.edu.cn).

Chunfu Jia and Fei Duan are with the College of Cyber Science, Nankai University, Tianjin 300350, China, and also with the Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China (e-mail: cfjia@nankai.edu.cn; duanf@mail.nankai.edu.cn).

Digital Object Identifier 10.1109/TIFS.2021.3107147

`password` as "reject" and "few character types", and it evaluates `password123` and `P@ssword` which are the most commonly used weak passwords as "accept" and "medium security". Attackers can easily crack these passwords with simple leet rules (see Hashcat [10] and fuzzyPSM [8]).

Except for fuzzyPSM [8], mainstream PSMs (e.g., [11]–[17]) in academia and industry all assume that passwords are newly constructed from scratch when a user registers, which doesn't conform to the real-world password construction habits: According to research on users' habits of constructing passwords [3], [8], about 80% of users show password reuse behaviors, and about half of users reuse passwords without modification. Thus, a well-designed PSM shall consider users' reuse behaviors, and fuzzyPSM [8] generally performs the best in evaluating passwords with low *guess numbers* accordingly.

FuzzyPSM [8] is based on the fuzzy-PCFG algorithm. PCFG [18] (i.e., probabilistic context-free grammar) is good at accurately capturing password structures based on character types, but it can only generate passwords with similar structures/parts to passwords in the training set [19]. PCFG-based PSM [16] also retains these properties. For fuzzyPSM, Wang *et al.* [8] took datasets of weak strength (e.g., Rockyou and Tianya) as base dictionaries, and took datasets of moderate strength (e.g., Phpbb and Weibo) as training sets. They considered six kinds of common leet transformations and the corresponding probabilities, including `a`↔`@`, `s`↔`$`, `o`↔`0`, `i`↔`1`, `e`↔`3` and `t`↔`7`. They also considered the probability of capitalizing the first letter of a base password segment. Therefore, fuzzyPSM [8] performs the best under online guessing [20], and can be improved under offline guessing scenarios.

Inspired by fuzzyPSM [8], we propose a robust and accurate PSM called RLS-PSM, based on password Reuse, Leet transformations and Separation. In contrast with fuzzyPSM [8], we mainly make the following improvements:

– We construct the basic dictionary, which is composed of basic passwords (consisting of lowercase letters and digits) in password datasets of the same service type with the targeted site. This is because users' password reuse behaviors often occur between accounts of similar service types [21]. Thus our method can obtain accurate parameter values when RLS-PSM is used for different service types. It is worth noting that, the "basic (base) passwords" of fuzzyPSM [8] are in a base dictionary (i.e., a password dataset of weak strength), which are quite different from our "basic passwords". See Sec. III-B for details.
– Based on password reuse, we divide users' password construction behaviors into capitalization in all positions of a password, and those that use special characters for leet transformations or separation. We, for the first time, propose *fuzzy matching algorithm* to calculate the probabilities of these usage cases.
– Considering that user-constructed passwords are usually simple in structures and rich in semantics, we use Long Short-Term Memory (LSTM) to calculate the probabilities of basic passwords. Furthermore, the composition of strings generated by LSTM is more random and diverse [9], [22], which makes RLS-PSM more robust than its counterparts under offline guessing scenarios (see Sec. IV-C.1). Moreover, RLS-PSM performs only slightly inferior to fuzzyPSM [8] in online guessing scenarios.

Most PSMs in the industry can only give users simple strength rating results (e.g., "strong", "medium", "weak") or strength color bars. Thus, we believe that a PSM will be better if it makes users have an intuitive sense of the strength of their passwords [5]. Hence RLS-PSM shows two aspects of password strength to users: *Absolute strength* and *relative strength*. Website administrators can set up *benchmark passwords* according to the method in Sec. III-C to show *relative strength*. Meanwhile, RLS-PSM also shows a user the ranking of her password strength in the system, that is, peer-pressure motivator [7], expressed as the top percent. For an entered password that is not strong enough, RLS-PSM will generate several groups of *candidate passwords* with sufficient strength and recommend them to the user.

The economic value and password policies of diverse services are quite different, with an apparent "stratification" phenomenon [23]. Password strength is not a constant and it largely depends on its application scenario [17]. Therefore, the parameter settings of PSMs shall be adjusted accordingly. Based on these facts, we select six kinds of common websites with different service types, and set parameters for RLS-PSM respectively. This makes RLS-PSM very practical in the following scenario: A newly established website can directly apply the robust RLS-PSM with parameters set under the corresponding service type. This is proved by the results of classification validity experiments in Sec. IV-C.1.

A. Contributions

(1) **A new PSM.** We propose a robust and accurate PSM, called RLS-PSM, based on password Reuse, Leet transformations and Separation. We use the Monte Carlo method [24] to simulate *evaluating attempts* to measure the robustness, and use the weighted Spearman correlation coefficient (wspearman) [20] to measure the accuracy. We compare RLS-PSM with leading PSMs (including Markov-based PSM [17], PCFG-based PSM [16], fuzzyPSM [8], RNN [15], and Zxcvbn [14]), and password cracking tools (including Hashcat [10] and JtR [25]). The experimental results show that, the robustness of RLS-PSM is significantly higher than its counterparts, especially in offline guessing scenarios (i.e., *evaluating attempts* $> 10^4$). For example, on average, Fraction of Successfully Evaluated (FSE) passwords of RLS-PSM is 18.9% higher than fuzzyPSM [8], and its average FSE reaches 83.1% when *evaluating attempts* $\approx 10^{10}$. The accuracy of RLS-PSM is also better than other mainstream PSMs used for comparison in this paper, except for fuzzyPSM [8].

(2) **Insights on password reuse.** We divide password construction behaviors based on password reuse into capitalization in all positions of a password, and those that use special characters for leet transformations or separation. We, for the first time, propose *fuzzy matching algorithm* to calculate the probabilities of these usage cases.

TABLE II
BASIC INFO ABOUT THE REAL-WORLD PASSWORD DATASETS

Service type	Dataset	Location	Language	When leaked	Total PWDs	Sum
Portal	Yahoo!	USA	English	July 2012	434,291	434,291
Commercial	Taobao	China	Chinese	Feb. 2016	14,900,108	29,742,646
	Dodonew	China	Chinese	Dec. 2011	14,842,538	
Email	126	China	Chinese	Oct. 2015	230,328	5,156,974
	Gmail	Russia	Hybrid†	Sep. 2014	4,926,646	
Game	178	China	Chinese	Dec. 2011	9,072,961	27,990,158
	7k7k	China	Chinese	Dec. 2011	18,917,197	
Forum	CSDN	China	Chinese	Dec. 2011	6,427,877	6,427,877
Social	Tianya	China	Chinese	Dec. 2011	29,513,716	73,135,657
	Renren	China	Chinese	Dec. 2011	3,257,831	
	Rockyou	USA	English	Dec. 2009	14,339,403	
	Twitter	USA	English	June 2016	26,024,707	

† The language is mainly in Russian.

With this method, we overcome the shortcomings of previous research, such as the lack of important transformations (e.g., JtR [25] only stipulates limited heuristic rules), and the inability to calculate probabilities (e.g., Zxcvbn [14]). What's more, seeing that user-constructed passwords are usually simple in structures and rich in semantics [26], [27], we adopt LSTM to calculate the probabilities of alphanumeric substrings of a password, which achieves a good balance between accuracy and efficiency.

(3) **Practical PSM configuration.** We, *for the first time*, use *benchmark passwords* in a PSM to show a user the *relative strength* of her password, and recommend sufficiently strong *candidate passwords* based on the entered password. This is recommended in Human-Computer Interaction research by Egelman *et al.* [7]. Taking into account that the economic value and password policies of different websites are varied, we select 12 large-scale real-world password datasets leaked from sites of six most common service types to train RLS-PSM respectively, and give the corresponding parameter settings. We also do homogeneous and heterogeneous experiments to prove the importance of setting different parameters of PSMs based on the service types.

II. EVALUATION FRAMEWORK

To accurately evaluate password strength, we need to comprehensively consider users' password construction behaviors. Before that, we show the basic information of 12 leaked password datasets of six service types used in this paper in Table II. The characteristics of each service type are shown in Sec. III-C and Sec. IV-A.

A. Impact of Character Types on Password Security

NIST PSM [11] divides characters in passwords into four types: Lowercase letters (L), uppercase letters (U), digits (D), and special characters (S). Because the passwords with short lengths and only basic characters (i.e., lowercase letters and digits) are easy to crack, website administrators often encourage users to set strong passwords, by specifying the minimum character types in passwords. In some ways, this method can improve password security [28]. In Tianya dataset (see Table II), we calculate the ranking of passwords after a series of common transformations on the basic password `password`, and test their actual *guess numbers* with Markov-based PSM [17], PCFG-based PSM [16] and Zxcvbn [14]. The results are shown in Table III.

TABLE III
GUESS NUMBERS OF PASSWORDS OBTAINED BY DIFFERENT TRANSFORMATION METHODS GIVEN BY FOUR TYPICAL PSMS

Typical PSM	Ranking†	M-3‡ [17]	M-4‡ [17]	PCFG‡ [16]	Zxcvbn§ [14]
`password`	8.00×10^{1}	2.29×10^{2}	8.10×10^{1}	7.40×10^{1}	3.00×10^{0}
`Password`	1.82×10^{4}	7.44×10^{3}	2.00×10^{3}	5.45×10^{3}	5.00×10^{0}
`Password!`	2.97×10^{5}	1.18×10^{7}	2.95×10^{6}	1.46×10^{7}	1.11×10^{4}
`password123`	9.60×10^{1}	1.84×10^{5}	2.61×10^{4}	2.03×10^{4}	5.96×10^{2}
`Password123`	4.91×10^{4}	3.86×10^{6}	4.52×10^{5}	4.86×10^{5}	1.19×10^{3}
`p@ssword123`	6.25×10^{6}	5.51×10^{6}	8.08×10^{6}	5.90×10^{11}	1.50×10^{4}
`P@ssword123`	5.64×10^{5}	4.96×10^{6}	7.68×10^{6}	3.63×10^{12}	1.50×10^{4}
`password@123`	7.98×10^{5}	1.10×10^{7}	1.13×10^{6}	5.56×10^{6}	1.01×10^{6}

† Ranking: The ranking of a password in descending order of frequency in Tianya dataset in Table II.
‡ M-3 and M-4: 3-gram and 4-gram Markov-based PSM [17]. PCFG: PCFG-based PSM [16].
§ It should be noted that, unlike PCFG-based PSM [16] and Markov-based PSM [17], the *guess numbers* output by Zxcvbn [14] is just a "score" that can reflect password strength.

It can be seen from Table III that, even if the most common transformations are adopted, such as initial capitalization (e.g., `password`→`Password`), appending special characters at the end of the password (e.g., `Password`→`Password!`), and changing some characters into special characters (e.g., `password123`→`p@ssword123`), *guess numbers* fluctuate significantly. Moreover, password strength can be substantially increased by adding special characters compared with initial capitalization (e.g., `password123`→`Password123` and `password123`→`password@123`). Another interesting phenomenon is that, when a password already contains special characters, initial capitalization may counter-intuitively reduce password strength (e.g., `p@ssword123`→`P@ssword123`), which may be caused by users' preferences. Therefore, we cannot simply think that capitalization or adding special characters can undoubtedly increase the password strength. To measure the impact of uppercase letters and special characters on password strength, we propose *fuzzy matching algorithm* for RLS-PSM (see Sec. III-B).

B. Capitalization and Leet Transformations

Some PSMs (e.g., [8], [14]) and password cracking tools (e.g., [10], [25]) do assess the impact of uppercase letters on password strength. Specifically, Zxcvbn [14] doesn't take into consideration where the uppercase letters appear, it estimates the capitalization factor of a password by measuring the proportion of uppercase letters. Hashcat [10] and JtR [25] set the rule that any lowercase letter in a password can be transformed to the corresponding uppercase letter. This rule could be better convinced if it calculates how likely a lowercase letter is to be transformed into an uppercase letter. Based on this, fuzzyPSM [8] considers the probability of capitalizing the first letter of a password. However, it seems necessary to take into account the uppercase letters that appear in other positions besides the first, because they account for a large proportion of certain service types (such as Portal

TABLE IV

TOP 10 PASSWORDS WITH SPECIAL CHARACTERS IN DATASETS OF DIFFERENT SERVICE TYPES†

Ranking	Portal	Commercial	Email	Game	Forum	Social
1	**123456..**	**123456.**	**3.1415926**	**123456.**	**3.1415926**	**123456.**
2	**123456789.**	**3.1415926**	**123456.**	**3.1415926**	**123456.**	***123456**
3	**p@ssw0rd**	**123456789.**	**1230.1231**	**1230.123**	**1230.123**	**123456..**
4	**.....**	**123456..**	**123456..**	**123456..**	**123456..**	**1230.123**
5	**123...**	**.....**	**123456789..**	**.....**	**123456789.**	**123456789.**
6	**123.123**	**123.123**	**p@ssw0rd**	**123456789.**	**.....**	**.....**
7	**0.123456**	**520.1314**	**.....**	**123...**	**123...**	**fashion@000**
8	**0.0.0.**	**abc_123**	**123...**	**123.123**	**123.123**	**benq*edifer**
9	**123456...**	**0.0.0.**	**123.123**	**0.0.0**	**0.123456**	**p@ssw0rd**
10	**123456789***	**123.123**	**0.123456**	**0.123456**	**0.0.0.**	**123.123**

† A password in bold **black** means that the special characters in it play the role of separating common strings of digits or letters, and are regarded as separators. A password in bold **blue** means that it has clear semantics (e.g., `3.1415926` is π, `.....` is an ellipsis), and the special characters in it are not obtained from digits or letters by leet transformations; we view them as separators. A password in bold **red** means that the special characters in it are obtained from digits or letters by leet transformations (e.g., `a`→`@`, `o`→`0`).

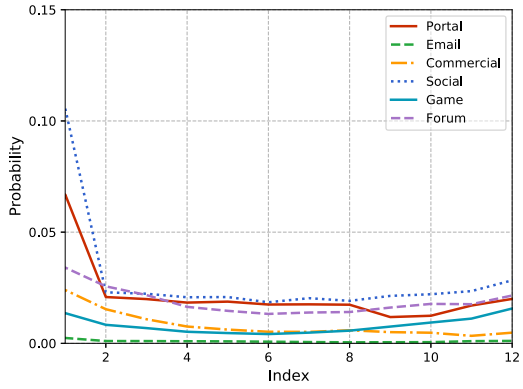


Fig. 1. Probabilities of capitalization in different positions of passwords of different service types (see Table II). The value of index represents the corresponding index position of the password.

and Social), as shown in Fig. 1. This is more in line with users' password construction behaviors. As the probabilities of different uppercase letters are not the same, we calculate them at any position of a password.

Many password cracking tools (e.g., JtR [25] and Hashcat [10]) preset dozens of leet transformation rules, including single character transformation (e.g., `a`→`@`) and multi-character transformation (e.g., `for`→`4`). These tools are heavily reliant on dictionaries and will get saturated soon, and have low robustness [19], [29]. Zxcvbn [14] specifies different transformations in each pattern. None of these PSMs give the probability of each leet transformation yet. FuzzyPSM [8] based on the fuzzy-PCFG algorithm specifies six common leet transformations. It could be better if it covers a wider variety of common leet transformations and the probabilities. So we propose *fuzzy matching algorithm* to find and record varied leet transformations and calculate the corresponding probabilities in password datasets of different service types.

C. Role of Special Characters in Passwords

In previous studies, special characters in passwords are often regarded as randomly selected strings (e.g., S segment in PCFG-based PSM [16]), common characters (e.g., Markov-based PSM [17]) or the results of performing leet transformations (e.g., `a`→`@` and `s`→`$` in fuzzyPSM [8]). However, Wang *et al.* [8] pointed out that a large proportion of users would choose to insert special characters in different positions of old passwords to improve their strength. Specifically, to facilitate memory, many users add special strings before (e.g., `*123456`), between (e.g., `1230.123`) or after (e.g., `123456..`) basic semantic strings when modifying old passwords, see Table IV. Hashcat [10] and JtR [25] also set rules for adding special characters in different positions of a password to enhance the ability to crack complex passwords. In Table IV, we show the top-10 passwords with special characters in datasets of different service types mentioned in Table II, and find that special strings in passwords often play the role of "separator". So we propose *fuzzy matching algorithm* to detect how special characters serve as separators and calculate the corresponding probabilities.

Based on the above considerations, for RLS-PSM, we select password datasets of different service types and account values, and then divide them into basic datasets and special character datasets, as the training sets, and give the parameter settings suitable for different application scenarios.

III. ALGORITHMS

A. Module Design

RLS-PSM consists of three modules: Preprocessing module, Probability calculation module, and Strength feedback mudule. We will introduce them separately below.

1) Preprocessing Module: To quantify the impact of special characters and uppercase letters on password strength, we set *Character transformation dictionary* to record the probabilities of the cases that each basic character remaining unchanged or transforming to an uppercase letter or a special character, and the probabilities that each special string as a separator. Through *fuzzy matching algorithm* (i.e., Algorithm 1), we can convert a password into a combination of basic strings and special strings served as separators.

We also set *Password structure dictionary* to record password structures and the corresponding probabilities. In a password, the substring that can be obtained from a basic password through capitalization or leet transformations is denoted as B, and its probability can be calculated by LSTM; and the special string of length n as a separator is denoted as S_n. Thus, the password structure can be expressed as $[\ \hat{}B?(S_nB)^*S_n?\$]$.

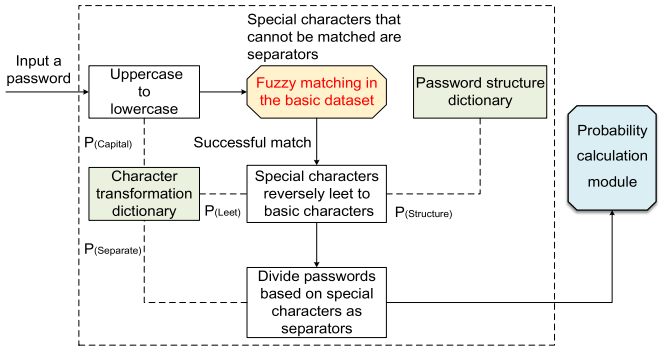


Fig. 2. Preprocessing module. *Character transformation dictionary* records the probabilities that each basic character remaining unchanged or transforming to an uppercase letter or a special character, and the probabilities that each special string as a separator. *Password structure dictionary* records password structures and the corresponding probabilities.

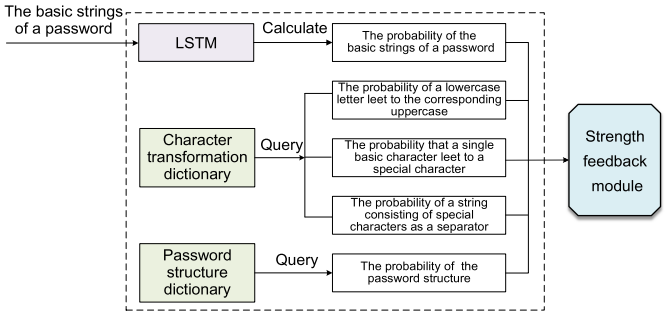


Fig. 3. Probability calculation module. *Character transformation dictionary* and *Password structure dictionary* have the same meanings as in Fig. 2.

For example, the structure of `P@ssword####123` is BS_4B. We use the Laplace smoothing suggested by Ma *et al.* [30] when calculating the probability of the password structure. See Sec. III-B for details. The process is shown in Fig. 2.

2) Probability Calculation Module: We have obtained the probabilities of capitalization, leet transformations and separators of the entered password in Preprocessing module. Then we use LSTM to calculate the probabilities of basic strings output by Preprocessing module. We can get the probability of the structure of the entered password from *Password structure dictionary* afterwards. By multiplying these probabilities, we can obtain the probability of the entered password. The process is shown in Fig. 3.

3) Strength Feedback Module: Although users can acquire password strength (i.e., the score or the probability provided by a PSM), it is likely that they don't understand the real security of a password. Egelman *et al.* [7] find that a user would get a better understanding of password strength, if a PSM also provides feedback about how her password compares to peers. Thus, we not only provide both *absolute strength* and *relative strength* of a password, but also implement the peer-pressure motivator proposed in [7]. *Absolute strength* reflects the actual strength of a password, and *relative strength* is the difference between *absolute strength* and the strength of a certain *benchmark password*. Website administrators using RLS-PSM can set their own *benchmark passwords* according to different application scenarios and security requirements.

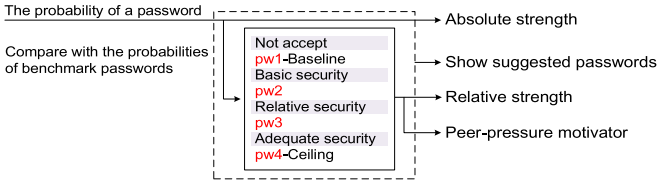


Fig. 4. Strength feedback module. `Pw1`, `pw2`, `pw3` and `pw4` are *benchmark passwords* to measure the *relative strength* of a password. Peer-pressure motivator proposed in [7] means the ranking of the strength of a user's password in the system, expressed as the top percent.

For *benchmark passwords*, we elaborate on their specific meanings, selection principles and recommended values in Sec. III-C. Here we only briefly describe the role of each *benchmark password*. Among *benchmark passwords*, `pw1` is the strength baseline. RLS-PSM rejects the password with the strength lower than `pw1`. `Pw2` is the boundary between basic security and relative security. `Pw3` is the boundary between relative security and sufficient security. A password with the strength between `pw1` and `pw3` can be accepted, but it is still not secure enough. Thus, RLS-PSM generates multiple *candidate passwords* for users to select based on the entered password. `Pw4` is the strongest password preset by the system. The process is shown in Fig. 4.

B. Dictionary Construction, Password Preprocessing and Probability Calculation

We construct dictionaries based on the training set and use them to calculate the probabilities of passwords in the test set. Firstly, we transform the uppercase letters in each password in the training set S to the corresponding lowercase letters, and get a new dataset S_{lower}. Then we count the frequency of each uppercase letter and basic character in S, and calculate the probability that each lowercase letter transforms into the uppercase letter and keeps unchanged. Secondly, we divide S_{lower} into two sets according to character compositions: Set S_{basic} consists of strings with only basic characters, and set $S_{special}$ consists of strings with basic characters and special characters. We need to set an appropriate confidence ratio (CR), and use *fuzzy matching algorithm* to perform leet matching for each password in $S_{special}$.

It is observed that a user often performs transformations on common basic passwords to get the final password, so we build the dataset S_{match}. This dataset consists of basic passwords with high frequencies in S_{basic}, generally *frequency*$\geq$10. When S_{match} is large enough, passwords with lower frequencies have little effect on the probabilities calculated by *fuzzy matching algorithm* (i.e., Algorithm 1). However, we haven't delved into the optimal frequencies in different cases, achieving the optimal balance between the accuracy and cost, thereby improving RLS-PSM. We treat this as a reserved issue for future resolution. Nevertheless, this setup is sufficient to support our analysis.

We record the string to be matched in $S_{special}$ as str_1, and try to match it with the string str_2 in S'_{match} in descending order of frequency. S'_{match} is a subset of S_{match} and satisfies $len_{(str_2\ in\ S'_{match})} \leq len_{str_1}$. For each str_2, the number of matching rounds is $len_{str_1} - len_{str_2} + 1$. We compare str_1 with

Algorithm 1 Fuzzy Matching Algorithm

Input: Entered password $PW = p_0p_1p_2p_3 \dots p_n$, $confidence\ ratio$, S_{match}
/* $len\,(PW) = n$ and p_i is the i-th char in the password PW. */
Output: Processed Password

```
for P ∈ S_match do
    P = c_0c_1c_2 ... c_m /* len(P) = m and c_i is the i-th char in the password P. */
    if m ≤ n then
        for offset ← 0 to (n − m + 1) do
            Fuzzy Counts ← 0
            for index ← 0 to m do
                if p_{index+offset} ∈ L ∪ D and p_{index+offset} ≠ c_index then
                    /* p_{index+offset} is the basic type. */
                    Break
                if p_{index+offset} ∈ S then
                    /* p_{index+offset} is the special type. */
                    Fuzzy Counts ++
                    Tag p_{index+offset}, Tag c_index
matching ratio ← 1 − Fuzzy Counts/(m + 1)
if matching ratio ≥ confidence ratio then
    Update password: tagged p_{index+offset} ← c_index
    Processed PW ← Updated PW
```

str_2 character by character. If the character to be matched in str_1 is a basic character, and it is different from the character in the corresponding position of str_2, we consider this round of matching fails. Then we move str_1 one character left and try again to match it with str_2. If the character to be matched in str_1 is a special character, we temporarily believe that this character matches the basic character in the corresponding position of str_2 successfully. Then we try to match the next character of str_1 and the next character of str_2.

When one round of matching is completed, we calculate the matching ratio (MR). MR is the proportion of the number of successfully matched basic characters (denoted as $count$) in str_2, i.e., $MR = count/len_{str_2}$. When $MR < CR$, we define that this round fails, that is, special characters cannot be obtained from basic characters by leet transformations. When str_1 and str_2 matches successfully, we perform reverse leet transformations on special characters in str_1, according to the characters in the corresponding positions of str_2. The meaning of successful matching is that, a substring of str_1 is the same as str_2, or some basic characters in str_2 are transformed into special characters by leet transformations. If all characters in str_1 are matched, we end the matching attempts for str_1. Otherwise, we try to match str_1 with the next str_2 in S'_{match}, and repeat the process.

After trying to match all str_1 in $S_{special}$, we record all matched cases and the corresponding probabilities in *Character transformation dictionary*, including all capitalization and cases that basic characters leet to special characters. For each unmatched special string, we regard it as a separator and store its probability in *Character transformation dictionary*.

To understand the above process intuitively, we show a toy example here. Suppose $CR = 0.5$ and a certain password is `P@ssword####12`. We first record the capitalization `p`→`P` and get the string str_1=`p@ssword####12` ($len_{str_1} = 14$). So S'_{match} = {`123456`, `password`, `000012`} satisfies $len_{(str_2\ in\ S'_{match})} \leq len_{str_1}$.

When str_2 = `123456` and $len_{str_2} = 6$, $len_{str_1} - len_{str_2} = 8$. In the first round of matching, we try to match `p@sswo` in str_1 and str_2. Firstly, we compare `p` in `p@sswo` and `1` in str_2. Since the `p` and `1` are different basic characters, this round of matching fails. In the second round of matching, we try to match `@sswor` (in str_1) and str_2. Since `@` in `@sswor` is a special character, we temporarily believe that `@` matches `1` in str_2 successfully. Then we compare the first `s` in `@sswor` with `2` in str_2, and find this matching fails for `s` and `2` are different basic characters. Repeat the process and we finally find that `####12` and str_2 fails to match.

Then, we switch to the next basic password in S'_{match}. Now str_2 = `password`, $len_{str_2} = 8$ and $len_{str_1} - len_{str_2} = 6$. In the first round of matching, we try to match `p@ssword` (in str_1) and str_2. Firstly, we compare `p` in `p@ssword` and `p` in str_2, and find they are the same. Then we compare `@` in `p@ssword` and `a` in str_2, and temporarily consider them to match. Next we compare the first `s` in `p@ssword` and the first `s` in str_2, and find they are the same. Repeat the process until the first round of matching ends. At this time, $count = 7$. We find that $MR_{\texttt{p@ssword}} = count/len_{str_2} = 7/8 = 0.875 > CR$, which means that `p@ssword` successfully matches str_2. Thus, we record the leet transformation `a`→`@`, and update str_1 immediately to `password####12`. In the second round of matching, we try to match `assword#` (in str_1) and str_2. Repeat the process and we finally find that `rd####12` and str_2 fail to match.

Finally, we switch to the last basic password in S'_{match}. This moment, str_2 = `000012`, $len_{str_2} = 6$ and $len_{str_1} - len_{str_2} = 8$. In the first round of matching, we try to match `passwo` (in str_1) and str_2. Firstly, we compare `p` in `passwo` and the first `0` in str_2, and find this round of matching fails. Repeat the process until we try to match `####12` and str_2 in the last round of matching, where $count = 2$. We find that $MR_{\texttt{####12}} = count/len_{str_2} = 2/6 \approx 0.33 < CR$, which means that `####12` fails to match str_2. Since we have exhausted S'_{match}, although some strings in `P@ssword####12` have not been matched by the basic passwords (i.e., str_2) in S'_{match}, we have completed the matching attempts on `P@ssword####12`. We record the capitalization `p`→`P`, the leet transformation `a`→`@`, the separator `####`, the basic strings `password` and `12`, and the password structure BS_4B, accordingly. For BS_4B, the first B represents the password fragment `P@ssword` converted from the basic string `password`, S_4 represents the separator `####`, and the second B represents the basic string `12`.

According to the security and accuracy requirements of the real-life application scenarios, CRs are adopted to adjust the contribution of special characters to password strength, and to determine whether special characters are used for leet

TABLE V
PROBABILITIES OF LEET TRANSFORMATIONS OF CERTAIN LETTERS UNDER DIFFERENT CONFIDENCE RATIOS IN TIANYA AND CSDN DATASETS

Confidence ratio	Tianya					CSDN				
	a→@	s→$	i→!	a→!	a→$	a→@	s→$	i→!	a→!	a→$
0.90	1.45×10^{-4}	1.20×10^{-5}	3.10×10^{-5}	0	0	9.12×10^{-4}	3.10×10^{-5}	0	0	0
0.75	2.42×10^{-3}	4.52×10^{-4}	6.73×10^{-4}	0	0	2.00×10^{-3}	1.45×10^{-4}	0	0	0
0.50	2.42×10^{-3}	1.46×10^{-3}	6.85×10^{-4}	0	0	2.66×10^{-3}	4.27×10^{-4}	0	0	0
0.25	2.66×10^{-3}	1.47×10^{-3}	6.85×10^{-4}	2.36×10^{-6}	0	2.66×10^{-3}	4.34×10^{-4}	2.47×10^{-7}	0	0
0.10	2.86×10^{-3}	1.57×10^{-3}	7.23×10^{-4}	2.36×10^{-6}	3.24×10^{-7}	2.69×10^{-3}	4.34×10^{-4}	2.47×10^{-7}	1.12×10^{-7}	1.76×10^{-7}

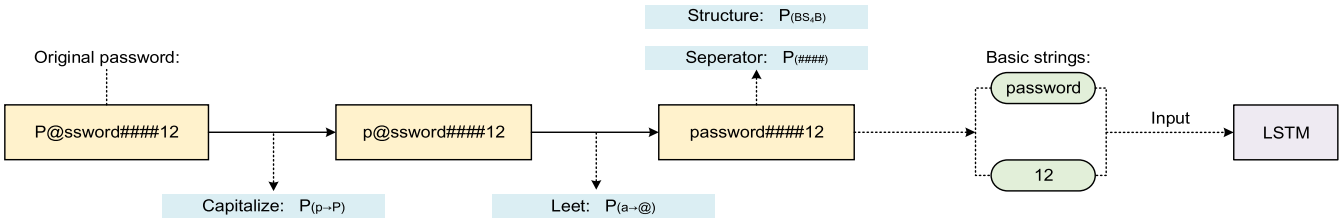


Fig. 5. Illustration of processing the entered password P@ssword####12.

transformations or separators. Table V shows the probabilities of leet transformations of certain letters under different CRs in Tianya and CSDN datasets (see Table II for details). The results show that the probabilities of leet transformations decrease with the increase of CRs. After building the dictionaries, we can calculate the probability of the entered password, based on the probabilities of different cases recorded in the dictionaries. Suppose the entered password contains basic characters, uppercase letters and special characters. First, we convert all uppercase letters in the password to the corresponding lowercase letters, and look up in *Character transformation dictionary* to get the transformation probabilities. Then we try to match the password with the basic passwords in S'_{match} using *fuzzy matching algorithm*, and record the cases that special characters in the entered password used for leet transformations or separators. We can also acquire the probabilities of these cases by looking up in *Character transformation dictionary*. Finally, we get the password structure (expressed as $[\hat{}B?(S_nB)^*S_n?\$]$, see Sec. III-A.1), and look up in *Password structure dictionary* to get the password probability. Each B in a password structure corresponds to a basic string, and we take all these basic strings as the input to LSTM to calculate their probabilities. For ease of understanding, we assume that the entered password is P@ssword####12 (same as the above example), and show the process in Fig. 5.

The LSTM settings adopted by RLS-PSM are one input layer, two hidden layers (256 neurons in each layer), one softmax layer and one output layer. After training on S_{basic}, LSTM can output the probability of the next character according to the prefix character sequence. LSTM carries out iterative operations continuously, and outputs the probabilities of all basic strings. With all probabilities obtained above, we can calculate the password probability by multiplying them. This process shows that RLS-PSM conforms to the probabilistic model. See Appendix for the proof.

C. Password Strength Feedback

We have introduced the design idea and basic concepts of Strength feedback module in Sec. III-A. The design principle of each *benchmark password* is elaborated below:

1) Pw1: guess numbers $\geq 10^4$. Network servers of modern service providers are equipped with the malicious traffic detection mechanism, and often limit the number of daily IP logins [19], [20], [23]. If the acceptable password cracking time for an attacker is four months, the maximum *guess number* of a large-scale online guessing is about 10^4 [23]. Since the most cost-effective method for the attacker is preferentially trying popular passwords to log in to the victim's account [31], many services adopt suspicious login detection [32] and lockout policies [33], which are effective to resist most online dictionary attacks. So we denote that pw1 is the baseline of password strength for websites. As long as a user's password is not in the blocklist, this basic strength can be achieved.

2) Pw2: guess numbers $\geq 10^7$. Suppose we ignore all the site restrictions, only consider the network transmission rate, and think that the targeted account is worth a large-scale guessing. Then an attacker's maximum *guess number* within the tolerable password cracking time is about 10^7 [32]. When *guess numbers* $> 10^7$, an attacker's profit will not increase significantly with the increase of *guess numbers* [23]. Thus, a password with pw2 strength can basically meet the requirements of most websites for password lengths and character compositions, and can also resist online guessing.

3) Pw3: guess numbers $\geq 10^{14}$. *Guess numbers* of offline guessing are much higher than that of online guessing, which can reach 10^{11} or even 10^{14} [13], [34]. Freeman *et al.* [32] has pointed out that the upper limit of *guess numbers* for offline guessing (breadth-first) is 10^{14}, and has proved that 10^{14} is the limit of the optimal profit in this case. Thus for pw3, we set *guess numbers* $\geq 10^{14}$, taking into account that *guess numbers* of most passwords are $10^6 \sim 10^{14}$. It can be seen that

TABLE VI
PROBABILITY INTERVALS OF BENCHMARK PASSWORDS OF RLS-PSM UNDER DIFFERENT SERVICE TYPES

Service	pw1	pw2	pw3	pw4
Portal	$[5.54\times10^{-8}, 1.36\times10^{-6}]$	$[5.42\times10^{-12}, 9.20\times10^{-11}]$	$[2.62\times10^{-20}, 3.39\times10^{-19}]$	$[5.93\times10^{-26}, 3.28\times10^{-25}]$
Commercial	$[2.04\times10^{-8}, 2.40\times10^{-6}]$	$[1.37\times10^{-12}, 2.97\times10^{-11}]$	$[1.93\times10^{-21}, 5.07\times10^{-20}]$	$[1.74\times10^{-28}, 9.80\times10^{-26}]$
Email	$[2.61\times10^{-8}, 9.67\times10^{-7}]$	$[2.62\times10^{-12}, 4.73\times10^{-11}]$	$[8.01\times10^{-21}, 1.08\times10^{-19}]$	$[6.98\times10^{-27}, 1.61\times10^{-25}]$
Game	$[1.79\times10^{-8}, 8.42\times10^{-7}]$	$[4.59\times10^{-12}, 5.09\times10^{-11}]$	$[1.30\times10^{-21}, 2.32\times10^{-18}]$	$[3.35\times10^{-27}, 7.82\times10^{-22}]$
Forum	$[6.06\times10^{-8}, 1.38\times10^{-6}]$	$[6.83\times10^{-12}, 1.20\times10^{-10}]$	$[1.55\times10^{-20}, 2.69\times10^{-19}]$	$[2.04\times10^{-29}, 3.35\times10^{-25}]$
Social	$[1.60\times10^{-8}, 5.47\times10^{-7}]$	$[1.14\times10^{-12}, 2.21\times10^{-11}]$	$[1.67\times10^{-21}, 2.17\times10^{-20}]$	$[1.56\times10^{-24}, 1.64\times10^{-21}]$

passwords with *guess numbers* $\geq 10^{14}$ can basically protect accounts against the existing GPU computing power [35].

4) Pw4: guess numbers $\geq 10^{20}$. We use the Monte Carlo method [24] to calculate the *guess numbers* and the corresponding coverage rates, and find that the coverage rates of all algorithms reach saturation when *guess numbers* are around 10^{20}. Thus passwords with *guess numbers* $\geq 10^{20}$ are secure.

Website administrators can adjust *benchmark passwords* according to the above principles. To reduce the cost of calculations, we transform the lower limit of *guess numbers* of a *benchmark password* into the corresponding probability interval, as shown in Table VI.

When the strength of the entered password is between pw1 and pw3, according to *Character transformation dictionary*, RLS-PSM recommends a number of strong enough *candidate passwords* for the user to choose from. RLS-PSM first randomly selects one or several characters of the entered password, and then randomly selects some transformation cases in *Character transformation dictionary* and execute them. These measures can ensure that *candidate passwords* are sufficiently secure (*guess numbers* $\geq$ pw3) and memorable.

IV. EXPERIMENTS

In this section, we first show the basic information about password datasets used in this paper. Then, we introduce mainstream PSMs and password cracking tools for comparison, and explain why they are selected. Next, we describe the experimental methods of robustness and accuracy analysis. Finally, we discuss the corresponding experimental results.

A. Password Datasets for Experiments

Stobert and Biddle [36] have shown that users are rational, and they often set passwords with high strength for high-value accounts and low strength for low-value accounts. This will result in an obvious stratified phenomenon in the strength of password datasets of different service types. Considering the economic value and password policies of websites, users often reuse passwords among similar accounts [3], [9], [26], [36]. Wang *et al.* [21] found that a single user may have many accounts on websites with service types such as shopping, email and forum, and she will reuse more passwords on these sites. To accurately evaluate password strength, we should regard the impact of different service types.

We collect 12 large-scale real-world password datasets leaked in the past ten years (see Table II), and they come from websites with different service types, scales, locations and languages. These datasets are all publicly available and are widely used in existing related research (e.g., [8], [15]–[17]). We divide them into the following six categories according to the security levels of different service types in [23].

Most portal accounts are mainly used for viewing, consulting and logging in to related sub-websites, so they belong to "don't care" category. Accounts of social networking websites are easy to retrieve even if they are compromised, thus they belong to "low-consequence" category. The compromise of game accounts will cause the decline of the game level or the loss of the virtual game equipment, but the impact is relatively limited. So they belong to "medium-consequence" category. The most commonly used email accounts are associated with many other websites and applications. If an email account is compromised, it will damage the security of a series of accounts, thus it belongs to "high-consequence" category. Commercial accounts are often related to various payment methods and personal privacy information. Such accounts will directly cause economic loss if they are compromised, and they belong to "ultra-sensitive" category, accordingly. For the forum category, we choose CSDN as the representative for the reason that its users are technicians with a high degree of password security awareness.

B. PSMs and Password Cracking Tools for Comparison

Unless otherwise specified, we will test the robustness and accuracy of the following PSMs and password cracking tools in Sec. IV-D and Sec. IV-E, and compare them with RLS-PSM.

1) PCFG-Based PSM / fuzzyPSM: In 2009, Weir *et al.* [18] proposed PCFG, which is recognized as a leading password attacking algorithm in academia. Based on this, in 2012, Houshmand and Aggarwal [16] applied PCFG [18] to password strength evaluation and designed PCFG-based PSM. In 2016, Wang *et al.* [8] firstly used the idea of password reuse for password strength evaluation, and proposed fuzzyPSM based on the fuzzy-PCFG algorithm. FuzzyPSM [8] performs the best in evaluating passwords with lower strength. To characterize users' password reuse behaviors, fuzzyPSM [8] requires a password dataset similar to that of the target website (corresponding to the test set) as a basic dictionary during training. Moreover, according to Wang *et al.*'s [8] suggestion, the basic dictionary should be weaker than the training set. Therefore, we choose the top-10^5 passwords of the training set as the basic dictionary for fuzzyPSM [8].

2) Markov-Based PSM: In 2012, Castelluccia *et al.* [17] put forward Markov-based PSM. In this paper, we choose 3-gram and 4-gram Markov-based PSM [17] for their good performance and acceptable cost.

3) RNN: Melicher *et al.* [15] proposed an RNN-based PSM in 2016. RNN [15] is deployed on the client side, allowing using special encoding and bloom filter, and has incomparable advantages over other existing PSMs in evaluating passwords with medium or higher strength.

4) JtR / Hashcat: Rule-based password cracking tools (e.g., JtR [25] and Hashcat [10]) can also be used to compare with PSMs. This is because *evaluating attempts* and *guess numbers* are obtained in similar ways. See Sec. IV-C for details. For JtR [25], we use the SpiderLabs mangling rules [37]. For Hashcat [10], we adopt the best64 and gen2 rule sets integrated in the software. Ur *et al.* [38] found that, as JtR [25] and Hashcat [10] guess in different orders, there are significant differences in the validity of guessing a certain password. JtR [25] first traverses the entire wordlist using the current rule, and then tries the next rule. On the contrary, Hashcat [10] first traverses all rules for one password, and then tries the next password, which requires higher computing power. We use Password Guessability Service (PGS) [38] to do the relevant experiments. Since passwords generated by JtR [25] and Hashcat [10] are not sorted, which doesn't meet the premise of accuracy test, we don't use them for accuracy comparison. Thus, we only use JtR [25] and Hashcat [10] for robustness test.

5) Zxcvbn: Wheeler [14] proposed an improved Zxcvbn in 2016. It is evaluated as the best pattern-detection based PSM [20], which quantifies the impact of various character types and patterns on password strength. In the assessment of Golla and Dürmuth [20], Zxcvbn [14] has no obvious disadvantages in accuracy compared with other PSMs based on attacking algorithms (e.g., [8], [15]–[17]). However, its *guess numbers* are usually not equal to the actual *guess numbers*. So Zxcvbn [14] is not suitable for evaluating the robustness. In addition, the Monte Carlo method [24] can be applied when the guessing probability is equal to the sampling probability. Apparently, Zxcvbn [14] doesn't meet this, so we only test its accuracy.

C. Experimental Methods of Robustness and Accuracy Analysis

1) Methods of Robustness Analysis: According to different design ideas, PSMs can be divided into rule-based (e.g., NIST [11]), pattern matching-based (e.g., Zxcvbn [14] and KeePass [12]), and attacking algorithm-based (e.g., RLS-PSM, PCFG-based PSM [16], fuzzyPSM [8], Markov-based PSM [17] and RNN [15]). Ur *et al.* [5] found that password guessability can be quantified as the time complexity of cracking a password, generally expressed as *guess numbers*, which is an effective measure of password strength. Therefore, attacking algorithm-based PSMs generally can better reflect the anti-guessing ability of a password, and can usually evaluate password strength more accurately and effectively compared to rule-based PSMs and pattern matching-based PSMs [8]. So we use *evaluating attempts* to measure the number of attempts (or cost) required for a PSM to successfully evaluate a password.

The evaluation process is as follows: First, a PSM generates a list of unique passwords in descending order of password probability. Then we try to match a password pw with passwords in the guess list. If pw matches the password with index N, it means that the PSM can evaluate the strength of pw under *N evaluating attempts*, and the password strength of pw is N. Otherwise, it indicates that the PSM cannot successfully evaluate the strength of pw under *N evaluating attempts*. In this paper, the robustness of a PSM refers to the ability of the PSM to evaluate password strength normally (i.e., without failure). Under the given *evaluating attempts*, the more passwords a PSM can successfully evaluate, the more robust the PSM is.

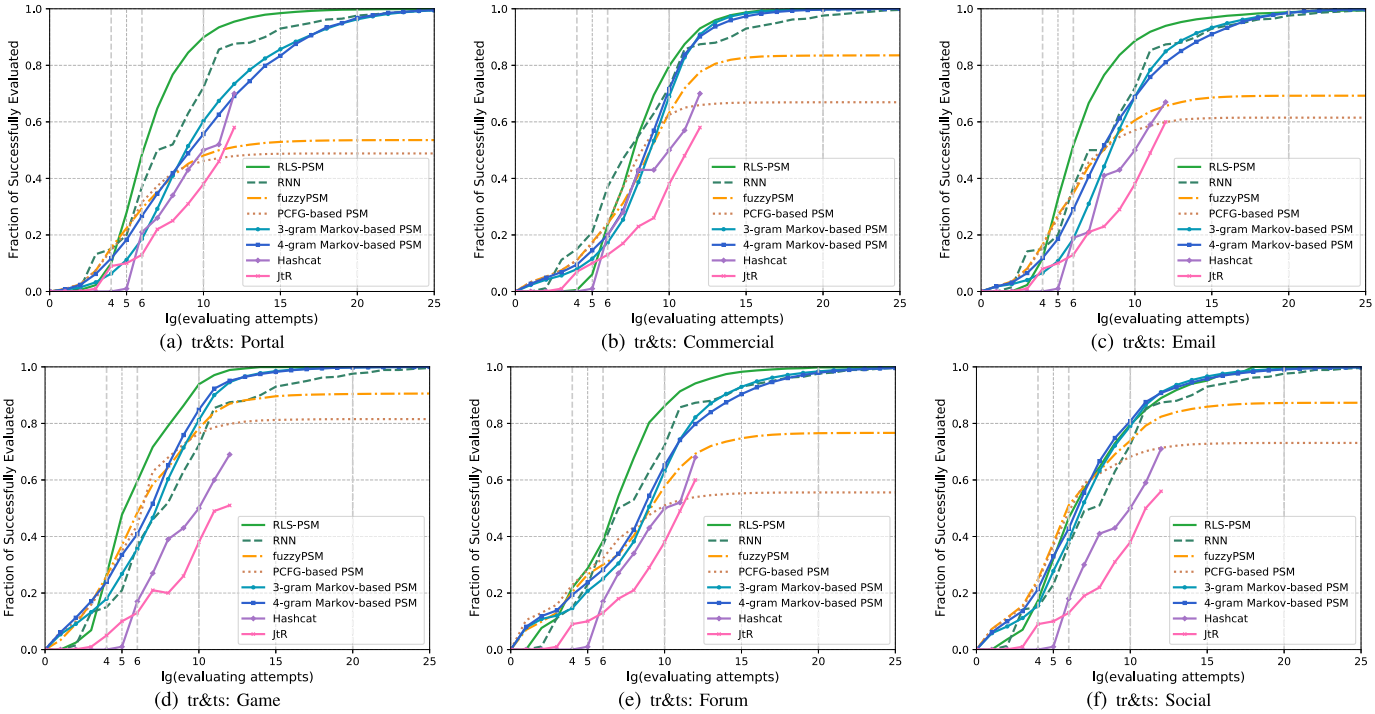
The robustness of an attacking algorithm-based PSM depends on the effectiveness of its built-in algorithm. Thus, we take the Fraction of Successfully Evaluated (FSE) to evaluate the robustness of a PSM. FSE means that, under a certain test set and *evaluating attempts*, the fraction of passwords that the PSM can successfully evaluate. Under the same *evaluating attempts*, the higher the FSE, the more passwords the PSM can successfully evaluate and the stronger the robustness. Since the methods of obtaining *evaluating attempts* and *guess numbers* are similar, password cracking tools like JtR [25] and Hashcat [10] can also be used to compare the robustness.

The robustness analysis includes three types of experiments: *Homogeneous experiments*, *heterogeneous experiments* and *classification validity experiments*. In all these experiments, $CR = 0.5$, and S_{match} is composed of passwords with *frequency* ≥ 10 in S_{basic}. For each PSM, after training the algorithm, we use the Monte Carlo method [24] to calculate the *evaluating attempts* for each password in the test set, and then draw the curve of FSE as *evaluating attempts* increase. The meanings of these three types of experiments are as follows, see Sec. IV-D for details.

a) Homogeneous experiments: The training set and test set of this kind of experiments have the same service type (e.g., tr: Portal, ts: Portal). Aside from Portal, for the passwords of each service type in Table II, we randomly select 1 million passwords as the test set, and use the remaining passwords as the training set. Since the number of passwords in Portal is relatively small, we randomly select half of the passwords as the training set and the other half as the test set. The experimental results are shown in Fig. 6.

b) Heterogeneous experiments: The training set and test set of this kind of experiments are of different service types (e.g., tr: Game, ts: Email). Other settings of heterogeneous experiments are the same as homogeneous ones. Due to space constraints, although we have done 30 groups of heterogeneous experiments, we only show 12 groups in this paper, as shown in Figs. 7 and 8. The unshown results are similar.

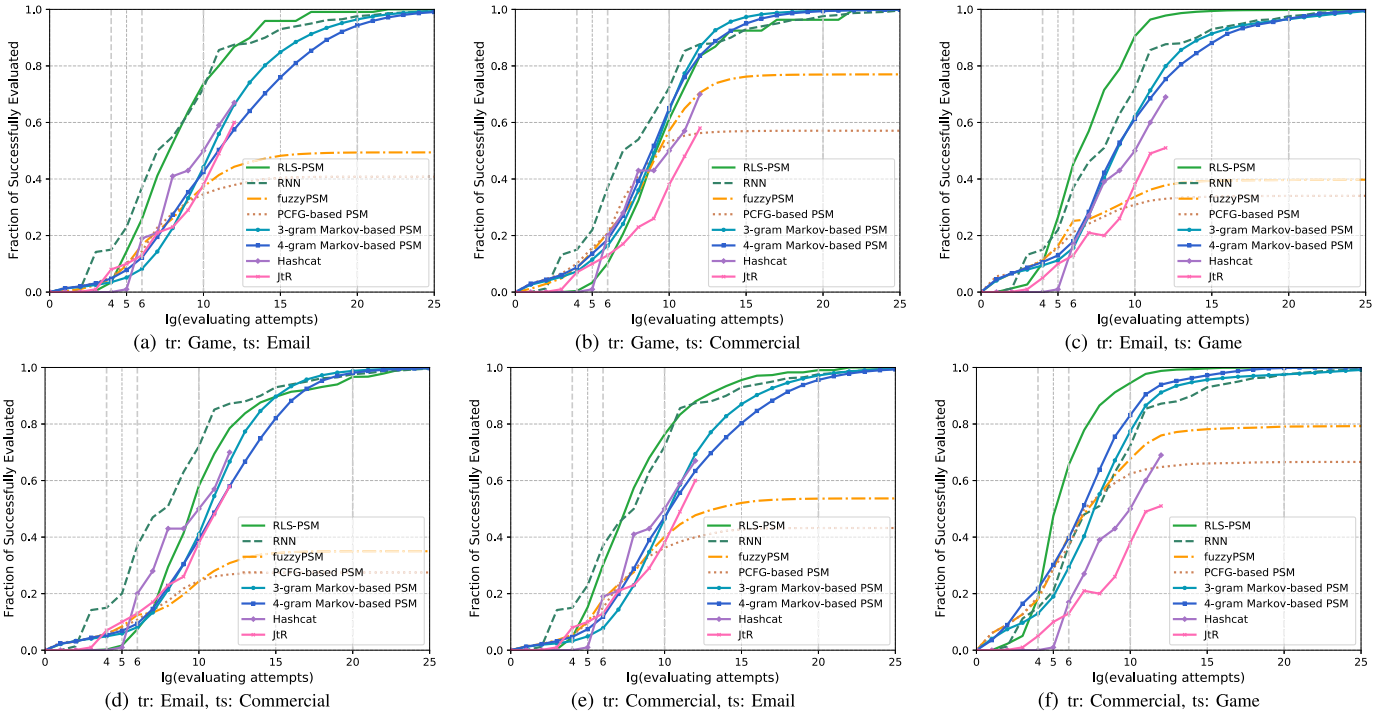
c) Classification validity experiments: The parameter setting of RLS-PSM is specific to the service type of the site, so the site manager can directly adopt RLS-PSM corresponding to the service type. Thus, classification validity experiments are to prove that, when trained with the password datasets of the same service type as the targeted website, the robustness of the PSM is almost the same



(a) tr&ts: Portal (b) tr&ts: Commercial (c) tr&ts: Email

(d) tr&ts: Game (e) tr&ts: Forum (f) tr&ts: Social

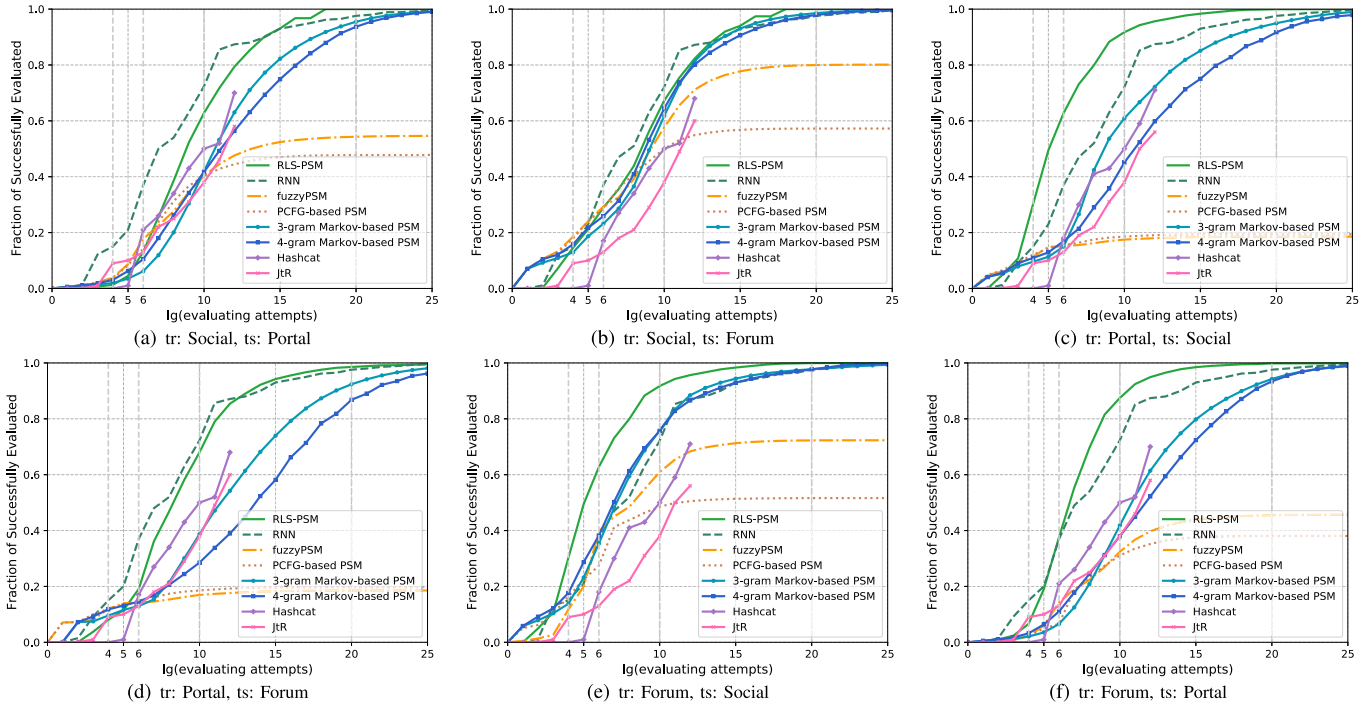
Fig. 6. Homogeneous experimental results for RLS-PSM, typical PSMs and password cracking tools. Homogeneous experiments are experiments with the same service type of the training set and the test set (e.g., tr: Portal, ts: Portal). The **green** lines show that RLS-PSM generally outperforms other mainstream PSMs when evaluating passwords with medium or high *evaluating attempts* ($>10^4$).



(a) tr: Game, ts: Email (b) tr: Game, ts: Commercial (c) tr: Email, ts: Game

(d) tr: Email, ts: Commercial (e) tr: Commercial, ts: Email (f) tr: Commercial, ts: Game

Fig. 7. Partial heterogeneous experimental results for RLS-PSM, typical PSMs and password cracking tools. Heterogeneous experiments are experiments with different service types of the training set and the test set (e.g., tr: Game, ts: Email). Due to space constraints, without loss of generality, here we only show results of the training set and test set with the service type of Game, Email or Commercial. The unshown results are similar. The results show that the performance of all peers is reduced to varying degrees compared to that in Fig. 6. While the **green** lines show that RLS-PSM generally outperforms other mainstream PSMs in most cases.



(a) tr: Social, ts: Portal (b) tr: Social, ts: Forum (c) tr: Portal, ts: Social

(d) tr: Portal, ts: Forum (e) tr: Forum, ts: Social (f) tr: Forum, ts: Portal

Fig. 8. Partial heterogeneous experimental results for RLS-PSM, typical PSMs and password cracking tools. Heterogeneous experiments are experiments with different service types of the training set and the test set (e.g., tr: Social, ts: Portal). Due to space constraints, without loss of generality, here we only show results of the training set and test set with the service type of Social, Portal or Forum. The unshown results are similar. The results show that the performance of all peers is reduced to varying degrees compared to that in Fig. 6. While the **green** lines show that RLS-PSM generally outperforms other mainstream PSMs in most cases.

as when trained with the targeted website's own password dataset.

Take the RLS-PSM training and testing on password datasets of Social as an example. We take a certain password dataset of Social as the targeted password dataset, denoted as TG; each other single password dataset is denoted as SG. We randomly select 1 million passwords in TG as the test set, and select the passwords as the training set in the following cases (denoted as C):

C1. Randomly select 1 million passwords from a certain SG as the training set.
C2. Randomly select 1 million passwords from the dataset obtained after merging all SG as the training set.
C3. Randomly select another 1 million passwords from TG as the training set.

The experimental results are shown in Fig. 9.

2) Methods of Accuracy Analysis: To evaluate the accuracy of a PSM, one can measure its distance from the ideal PSM.[1] This can be achieved by calculating the correlation between the ranked *guess numbers* produced by each meter [8]. Following Golla and Dürmuth [20], we use the weighted Spearman correlation coefficient (wspearman) to measure the accuracy of PSMs. As this indicator has been demonstrated to be one of the most reliable correlation metrics [20]. Note that most passwords appear infrequently, so that the strength of such a password is not suitable to be approximated by its frequency [8], [19], [39]. To reduce approximation errors, we select top-10^4 most popular passwords from different service types in Table II in the experiments, referring to the methods in [20] and [29].

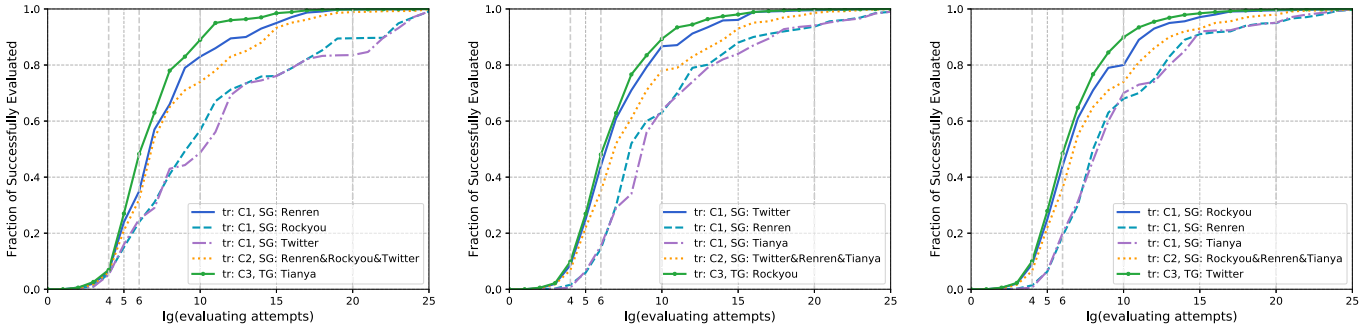
We use wspearman to measure the accuracy of 3-gram and 4-gram Markov-based PSM [17], PCFG-based PSM [16], fuzzyPSM [8], Zxcvbn [14], RNN [15] and RLS-PSM, respectively. Due to space constraints, we only show the detailed experimental results of Social, Email and Forum, as shown in Fig. 10. The unshown results are similar.

D. Robustness Analysis and Discussion

Modern network service providers tend to limit the password length and character composition, or adopt the blacklist detection mechanism to urge users to set more secure passwords. In this case, *evaluating attempts* of passwords are often greater than 10^4 [19], [23]. Many attacking algorithm-based PSMs (e.g., [15]–[17]) have the highest robustness when *evaluating attempts* are in $10^4 \sim 10^{10}$. PSMs are often limited by equipment, power and time costs, so usually *evaluating attempts* $< 10^{10}$. Hence, it is very important to effectively evaluate passwords with moderate strength (*evaluating attempts* in $10^4 \sim 10^{10}$). Fig. 6 shows that, in homogeneous experiments, RLS-PSM performs the best when evaluating passwords with medium or high *evaluating attempts* ($> 10^4$), which is significantly better than all mainstream counterparts for comparison.

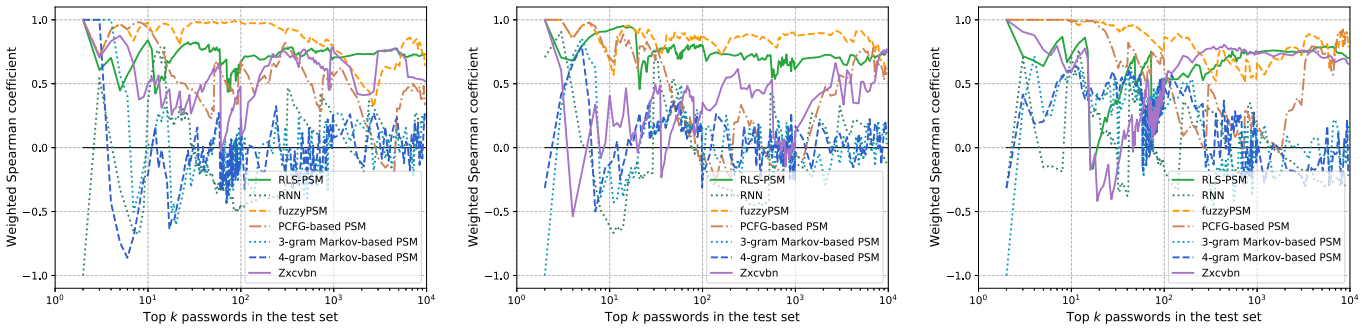
In homogeneous experiments, the corresponding FSE of RLS-PSM is about 15% and up to 20% when *evaluating attempts* $\approx 10^4$, and the FSE is more than 80% and up to 90% when *evaluating attempts* $\approx 10^{10}$. What's more, the social category contains many password datasets with

[1]The meaning of the ideal PSM is to use the frequency of a password in a real-world dataset to indicate password strength. The higher the frequency of a password, the lower the strength [8].



(a) ts (TG): Tianya (b) ts (TG): Rockyou (c) ts (TG): Twitter

Fig. 9. Partial classification validity experimental results. The unshown results are similar. These experiments are to prove that, when trained with the password datasets of the same service as the targeted website, the robustness of the PSM is almost the same as when trained with the targeted website's own password dataset. Here we take the RLS-PSM training and testing on datasets of Social as an example. We take a certain dataset of Social as the targeted password dataset, denoted as TG; each other single dataset is denoted as SG. As for different training and testing cases (i.e., C1, C2 and C3), see Sec. IV-C.



(a) tr&ts: Social (b) tr&ts: Email (c) tr&ts: Forum

Fig. 10. Accuracy experimental results of Social, Email and Forum for RLS-PSM and representative PSMs. A value k on the x axis means that the data point in the plot is calculated for the set of the top 1, 2,…, k-ranked passwords. The **green** lines indicate that, for accuracy, RLS-PSM is second only to fuzzyPSM [8] and is significantly better than other PSMs for comparison.

different languages and password policies. This is an important reason why RLS-PSM performs slightly worse in Social than in other service types. Moreover, this phenomenon also proves that in addition to the service type, the language and password policy of the training set can truly affect the experimental results. As for JtR [25] and Hashcat [10], their FSE is significantly lower than other PSMs when *evaluating attempts* are in $10^0 \sim 10^5$. But their FSE grows rapidly when *evaluating attempts* are in $10^5 \sim 10^{12}$. This may be due to the low overlap between their dictionaries and test sets.

Fig. 6, Fig. 7 and Fig. 8 show that, FSE of RLS-PSM grows rapidly when *evaluating attempts* are between 10^6 and 10^{20}, from about 50% to 99.9%, with a relative growth rate of about 100%. However, FSE of RLS-PSM under $10^0 \sim 10^4$ is slightly lower than some PSMs (e.g., fuzzyPSM [8] and RNN [15]). This is because most passwords with lower *evaluating attempts* are shorter in length and higher in frequency. So Markov-based PSM [17] and PCFG-based PSM [16] based on probability statistics can extract the connections between characters of such passwords quite efficiently. Thus their FSE is naturally higher. Nevertheless, RLS-PSM based on LSTM is not suitable for evaluating such passwords.

In Fig. 6, the saturation points of PCFG-based PSM [16] and fuzzyPSM [8] are in 50%∼83.1%, which are much lower than that of other PSMs (close to 100%). The reason is that the PSMs (such as [8], [16]) based on PCFG limit the number of generated password structures. With the increase of *evaluating attempts*, the number of password structures tends to be saturated, and password compositions tend to be single, so FSE almost stops growing. The saturation point of fuzzyPSM [8] is higher than that of PCFG-based PSM [16], since the structures of passwords generated by the former are based on basic strings, which to some extent increases the diversity of password compositions.

Because passwords with high *evaluating attempts* often have complex structures and various compositions, RLS-PSM based on LSTM performs quite well when *evaluating attempts* are higher, by generating a variety of random strings. For example, when *evaluating attempts* are in $10^6 \sim 10^{10}$, FSE of RLS-PSM is 13.9%∼36.3% higher than that of 3-gram Markov-based PSM [17], and 10.7%∼35.6% higher than that of 4-gram Markov-based PSM [17]. Even in heterogeneous experiments (see Fig. 7 and Fig. 8), when *evaluating attempts* $> 10^5$, RLS-PSM and RNN [15] based on LSTM have significantly higher FSE compared to other PSMs in most cases.

To efficiently evaluate the password strength, it is necessary to consider the service type of a password dataset. Except for Zxcvbn [14], most PSMs extract character compositions of password datasets during training, and some PSMs extract character transformations, too. When using a training set that has a different service type from the test set, the performance of PSMs will decrease significantly. It can be seen from Fig. 7

Fig. 8 that the performance of all PSMs decreases in different degrees in heterogeneous experiments, reflected in the slow growth of FSE and the decrease of saturation points. FSE of RLS-PSM is reduced by at least 5% when *evaluating attempts* $> 10^8$.

Here is an interesting phenomenon in heterogeneous experiments. That is, no matter what the service type of the training set is, when the service type of the test set is portal, the performance of RLS-PSM is always better than in other situations. There are two possible explanations. First, the test set of Portal has a larger proportion of passwords containing special characters and uppercase letters, which is more beneficial to *fuzzy matching algorithm*. These proportions are 10.5% for Portal, 5.7% for Email, 8.4% for Forum, 4.9% for Game, 19.6% for Social and 5.8% for Commercial. Second, the size of the test set of Portal (about 0.2 million) is significantly smaller than that of other service types (1 million), so FSE of Portal rises faster.

For classification validity experiments, without loss of generality, we take Fig. 9(a) with TG as Tianya as an example for analysis. For cases C1 (i.e., the training set is a dataset different from the test set) and C2 (i.e., the training set includes several datasets of the same service type), when *evaluating attempts* are 10^6 and 10^{20}, FSE can reach 37.3% and 82.3%. For case C3 (i.e., the training set is the same as the test set), when *evaluating attempts* are 10^6 and 10^{20}, FSE are 44.3% and 82.5%. This is because passwords in the same dataset are the most similar. On average, when SG is Renren, Rockyou and Twitter, FSE of case C1 is 3.1%, 14.2% and 14.5% lower than case C3, respectively. This shows that password datasets with the same language have more similar properties. Compared with case C1 (SG: Rockyou or SG: Twitter), FSE of case C2 increases by about 7%. Therefore, when there is a lack of sufficient training set, it is only required that the service type of the training set and the targeted website are the same; this can ensure a PSM has high robustness. The above analysis also highlights the influences of languages on passwords, and we will consider them in future work.

E. Accuracy Analysis and Discussion

The larger the wspearman value (ranges in [-1,1]) between the tested PSM and the ideal PSM, the better the accuracy of the former. In Fig. 10, we can see that, RLS-PSM maintains the advantage of fuzzyPSM [8] in terms of accuracy, and it is significantly better than other attacking algorithm-based PSMs for comparison (such as Markov-based PSM [17], PCFG-based PSM [16] and RNN [15]) and Zxcvbn [14]. It is worth pointing out that Zxcvbn [14], as a representative of pattern matching-based PSMs, is more accurate than RNN [15] and Markov-based PSM [17]. One possible explanation is that, popular passwords often contain many common patterns [8] (e.g., digit/letter sequences, keyboard patterns), which conforms to the design principle of Zxcvbn [14].

V. Conclusion

We have proposed a new password strength meter (PSM), called RLS-PSM, based on password reuse, leet transformations, and separation. We compared it with several mainstream PSMs (including Markov-based PSM [17], PCFG-based PSM [16], fuzzyPSM [8], RNN [15] and Zxcvbn [14]) and password cracking tools (including Hashcat [10] and JtR [25]) in terms of robustness and accuracy. The experimental results showed that, RLS-PSM generally significantly outperforms all its counterparts under offline guessing, and it is more accurate than other mainstream PSMs for comparison except fuzzyPSM [8]. In addition, RLS-PSM uses *benchmark passwords* to show a user the *relative strength* of her entered password and gives her *candidate passwords* for selection based on the original password. Furthermore, we did homogeneous and heterogeneous experiments respectively to reveal that, it is important to set different parameter values for a PSM under varied application scenarios.

Appendix
Proof That RLS-PSM Conforms to the Probabilistic Model

A PSM that complies with the probabilistic model can conveniently estimate the *guess number* of the password using the Monte Carlo method [24], and then calculate the password strength. In this way, there is no need to consume a large computing resources to generate the guess list to calculate the *guess number*. PCFG-based PSM [16], fuzzyPSM [8], Markov-based PSMs (such as 3-gram and 4-gram Markov [17]) and neural network-based PSMs (such as RNN [15]) all satisfy the probabilistic model. We use Υ to denote the set of characters that can be used in passwords, which usually includes 95 printable ASCII characters, namely lowercase letters (L), uppercase letters (U), digits (D), and special characters (S). Among them, L and D are basic characters (see Sec.II-A). Assuming that the password length l allowed by the website ranges from l_{min} to l_{max}, thus, the set of allowed passwords (denoted as Γ) is

$$\Gamma = \bigcup_{l=l_{min}}^{l_{max}} \Upsilon^l \tag{1}$$

A PSM based on the attacking algorithm calculates the construction probability of a password pw to evaluate its strength. When the sum of the construction probabilities of all passwords in Γ under a certain PSM is 1, it indicates that the PSM satisfies the probabilistic model, which is expressed as

$$P(\Gamma) = \sum_{pw \in \Gamma} P(pw) = 1 \tag{2}$$

RLS-PSM first converts uppercase letters in a password into the corresponding lowercase letters, and then divides the password into basic strings and special strings according to character types. A basic string (denoted as str_b) is composed of basic characters α, where $\alpha \in L \cup D$; STR_B is the set of basic strings. A special string of length i ($1 \leq i \leq l_{max}$) is denoted as str_s^i, which is composed of special characters β, where $\beta \in S$; STR_S^i is the set of such special strings. So there are

$$STR_B = \bigcup_{l=1}^{l_{max}} \alpha^l \tag{3}$$

and

$$STR_S^i = \bigcup \beta^i \tag{4}$$

We use B and S_i to mark basic strings and special strings of length i in passwords. Let $B\#$ be the password structure with B as the head and # as the remaining part; $S_i\#$ be the password structure with S_i as the head and # as the remaining part. The following equation can be obtained showing that the password structure conforms to the probabilistic model

$$P_{stru.}(B\#) + \sum_{i=1}^{l_{max}} P_{stru.}(S_i\#) = 1 \tag{5}$$

where $P_{stru.}(\cdot)$ means the probability of password structure.

RLS-PSM uses LSTM to calculate the probabilities of basic strings. According to the characteristics of LSTM, the sum of the probabilities of all basic strings is 1, so there is

$$\sum_{str_b \in STR_B} P(str_b) = 1 \tag{6}$$

The sum of the probabilities of special strings of the same length is also 1, which is expressed as

$$\sum_{str_s^i \in STR_S^i} P(str_s^i) = 1 \tag{7}$$

For RLS-PSM, some passwords do not contain uppercase letters, and special strings in them are used as separators. For example, the password `password%%%123` does not contain uppercase letters, and `%%%` is used as a separator. Meanwhile, `password` is the head, marked as B; `%%%123` is the remaining part, marked as #. In #, `%%%` is marked as S_3 and `123` is marked as B. For such passwords, the probability of the # part in Eq. 5 is expressed as

$$P(\#) = \prod_{B \in \#} \sum_{str_b \in STR_B} P(str_b) \times \prod_{S_i \in \#} \sum_{str_s^i \in STR_S^i} P(str_s^i) \tag{8}$$

Therefore, when the password structure is $B\#$, the sum of the password probability is

$$P(B\#) = P_{stru.}(B\#) \times \left(\sum_{str_b \in STR_B} P(str_b) \right) \times P(\#) \tag{9}$$

When the password structure is $S_i\#$, the sum of the password probability is

$$P(S_i\#) = P_{stru.}(S_i\#) \times \left(\sum_{str_s^i \in STR_S^i} P(str_s^i) \right) \times P(\#) \tag{10}$$

So the sum of the probabilities of the passwords that have not been transformed is

$$P(\Gamma') = P(B\#) + \sum_{i=1}^{l_{max}} P(S_i\#) = 1 \tag{11}$$

where Γ' is the set of passwords consisting of basic strings and separators.

However, other passwords contain uppercase letters or special characters that can be transformed from basic characters. For example, `P@ssword%%%123` is obtained by transformations `p`→`P` and `a`→`@` from the above `password%%%123`. So we record a basic character that can be transformed as δ, where $\delta \in L \cup D$. We mark a character transformed from δ as ϵ, and the corresponding transformation number as $n_{\delta \to \epsilon}$; the set of all possible ϵ is Θ, where $\epsilon \in \Theta$ and $\Theta \subset S \cup U$. Then the probability that δ transforms into ϵ is

$$P(\delta \to \epsilon) = n_{\delta \to \epsilon} / \sum_{\theta \in \Theta} n_{\delta \to \theta} \tag{12}$$

Record the probability of the character δ being transformed to other possible characters as $P_{trans.}(\delta)$, which satisfies

$$P_{trans.}(\delta) = \sum_{\theta \in \Theta} P(\delta \to \theta) = 1 \tag{13}$$

Record the set of basic characters that can be transformed as Δ, so there is

$$P(\Gamma) = P(\Gamma') \times \prod_{\delta \in \Delta} P_{trans.}(\delta) = 1 \tag{14}$$

The result of Eq. 14 is the same as that of Eq. 2, so RLS-PSM conforms to the probabilistic model.

ACKNOWLEDGMENT

The authors would like to thank the Passwords Research Team at Carnegie Mellon University for their password guessability service (https://pgs.ece.cmu.edu/), which was used in part of the experiments.

REFERENCES

[1] C. Herley and P. C. van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Security Privacy*, vol. 10, no. 1, pp. 28–36, Jan./Feb. 2012.

[2] S. Oesch and S. Ruoti, "That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers," in *Proc. USENIX SEC*, 2020, pp. 2165–2182.

[3] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proc. NDSS*, 2014, pp. 23–26.

[4] S. Yang, S. Ji, and R. Beyah, "DPPG: A dynamic password policy generation system," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 3, pp. 545–558, Mar. 2018.

[5] B. Ur *et al.*, "How does your password measure up? The effect of strength meters on password creation," in *Proc. USENIX SEC*, 2012, pp. 65–80.

[6] J. Tan, L. Bauer, N. Christin, and L. F. Cranor, "Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements," in *Proc. ACM CCS*, Oct. 2020, pp. 1407–1426.

[7] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, and C. Herley, "Does my password go up to eleven: The impact of password meters on password selection," in *Proc. ACM CHI*, Apr. 2013, pp. 2379–2388.

[8] D. Wang, D. He, H. Cheng, and P. Wang, "FuzzyPSM: A new password strength meter using fuzzy probabilistic context-free grammars," in *Proc. IEEE/IFIP DSN*, Jun. 2016, pp. 595–606.

[9] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE SP*, May 2019, pp. 417–434.

[10] J. Steube. (2018). *Hashcat*. [Online]. Available: https://hashcat.net/hashcat/

[11] W. Burr, D. Dodson, R. Perlner, S. Gupta, and E. Nabbus, "Electronic authentication guideline," NIST, Reston, VA, Tech. Rep. NIST SP800-63-2, Aug. 2013.

[12] D. Reichl. (Jul. 2015). *KeePass, Version 1.26/2.23: Password Quality Estimation*. [Online]. Available: http://keepass.info/help/kb/pw_quality_est.html

[13] M. Weir, S. Aggarwal, M. Collins, and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," in *Proc. ACM CCS*, 2010, pp. 162–175.
[14] D. L. Wheeler, "zxcvbn: Low-budget password strength estimation," in *Proc. USENIX SEC*, 2016, pp. 157–173.
[15] W. Melicher *et al.*, "Fast, lean, and accurate: Modeling password guessability using neural networks," in *Proc. USENIX SEC*, 2016, pp. 175–191.
[16] S. Houshmand and S. Aggarwal, "Building better passwords using probabilistic techniques," in *Proc. ACSAC*, 2012, pp. 109–118.
[17] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive password-strength meters from Markov models," in *Proc. NDSS*, 2012, pp. 1–14.
[18] M. Weir, S. Aggarwal, B. D. Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. IEEE SP*, May 2009, pp. 391–405.
[19] J. Galbally, I. Coisel, and I. Sanchez, "A new multimodal approach for password strength estimation—Part I: Theory and algorithms," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 12, pp. 2829–2844, Dec. 2017.
[20] M. Golla and M. Dürmuth, "On the accuracy of password strength meters," in *Proc. ACM CCS*, Oct. 2018, pp. 1567–1582.
[21] C. Wang, S. T. K. Jan, H. Hu, and G. Wang, "Empirical analysis of password reuse and modification across online service," 2017, *arXiv:1706.01939*. [Online]. Available: http://arxiv.org/abs/1706.01939
[22] Y. Cheng, C. Xu, Z. Hai, and Y. Li, "DeepMnemonic: Password mnemonic generation via deep attentive encoder-decoder model," 2020, *arXiv:2006.13462*. [Online]. Available: http://arxiv.org/abs/2006.13462
[23] D. Florêncio, C. Herley, and P. C. van Oorschot, "An administrator's guide to internet password research," in *Proc. USENIX LISA*, 2014, pp. 35–52.
[24] M. Dell'Amico and M. Filippone, "Monte Carlo strength evaluation: Fast and reliable password checking," in *Proc. ACM CCS*, Oct. 2015, pp. 158–169.
[25] A. Peslyak. (2019). *John the Ripper Community Build (1.9.0-Bleeding Jumbo)*. [Online]. Available: http://www.openwall.com/john/
[26] D. Wang, P. Wang, D. He, and Y. Tian, "Birthday, name and bifacial-security: Understanding passwords of Chinese web users," in *Proc. USENIX SEC*, 2019, pp. 1537–1555.
[27] R. Veras, C. Collins, and J. Thorpe, "On semantic patterns of passwords and their security impact," in *Proc. NDSS*, 2014, pp. 1–16.
[28] P. G. Kelley *et al.*, "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *Proc. IEEE SP*, May 2012, pp. 523–537.
[29] D. Pasquini, G. Ateniese, and M. Bernaschi, "Interpretable probabilistic password strength meters via deep learning," in *Proc. ESORICS*, 2020, pp. 502–522.
[30] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE SP*, May 2014, pp. 689–704.
[31] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2776–2791, Nov. 2017.
[32] D. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto, "Who are you? A statistical approach to measuring user authenticity," in *Proc. NDSS*, 2016, pp. 21–24.
[33] M. Alsaleh, M. Mannan, and P. C. van Oorschot, "Revisiting defenses against large-scale online password guessing attacks," *IEEE Trans. Depend. Sec. Comput.*, vol. 9, no. 1, pp. 128–141, Jan./Feb. 2012.
[34] M. L. Mazurek *et al.*, "Measuring password guessability for an entire university," in *Proc. ACM CCS*, 2013, pp. 173–186.
[35] D. Goodin, "Why passwords have never been weaker-and crackers have never been stronger," Ars Tech., Tech. Rep., 2012. [Online]. Available: https://arstechnica.com/information-technology/2012/08/passwords-under-assault/
[36] E. Stobert and R. Biddle, "The password life cycle," *ACM Trans. Privacy Secur.*, vol. 21, no. 3, pp. 13:1–13:32, 2018.
[37] The SpiderLabs. (2012). *Spiderlabs/Korelogic-Rules*. [Online]. Available: https://github.com/SpiderLabs/KoreLogic-Rules
[38] B. Ur *et al.*, "Measuring real-world accuracies and biases in modeling password guessability," in *Proc. USENIX SEC*, 2015, pp. 463–481.
[39] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proc. IEEE SP*, May 2012, pp. 538–552.

Qiying Dong received the bachelor's degree from Nankai University, China, in 2018, where she is currently pursuing the Ph.D. degree. She has received The Google Anita Borg Memorial Scholarship and The Cyber Security Scholarship, China. Her research interests include password security, web security, and authentication.

Chunfu Jia received the Ph.D. degree in engineering from Nankai University, China, in 1996. He has finished his Post-Doctoral Research with the University of Science and Technology of China. He is currently a Professor at Nankai University. His current interests include computer system security, network security, trusted computing, and malicious code analysis.

Fei Duan received the bachelor's degree from Nankai University, China, in 2021, where he is currently pursuing the master's degree. His research interests include password security, authentication, and machine learning.

Ding Wang received the Ph.D. degree in information security from Peking University, in 2017. From 2017 to 2019, he was supported by the "Boya Post-doctoral Fellowship" at Peking University. He is currently a Full Professor at Nankai University. He has published more than 60 papers at venues like IEEE S&P, ACM CCS, NDSS, Usenix Security, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING (TDSC), and IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY (TIFS). His research interests focus on passwords, authentication, and provable security. His research has been reported by over 200 medias like *Daily Mail*, *Forbes*, *IEEE Spectrum*, and *Communications of the ACM*, appeared in the Elsevier 2017 "Article Selection Celebrating Computer Science Research in China," and resulted in the revision of the authentication guideline NIST SP800-63-2. He has been involved in the community as the PC Chair/a TPC Member for over 60 international conferences, such as PETS 2022, ACSAC 2021/2020, ACM AsiaCCS 2022/2021, IEEE CNS 2020, IFIP SEC 2018–2021, and INSCRYPT 2018–2020. He has received the "ACM China Outstanding Doctoral Dissertation Award," the Best Paper Award at INSCRYPT 2018, and the First Prize of Natural Science Award of the Ministry of Education.