RANKGUESS: Password Guessing Using Adversarial Ranking

Tao Yang, Ding Wang
College of Cyber Science, Nankai University, Tianjin 300350, China; wangding@nankai.edu.cn
Key Laboratory of Data and Intelligent System Security (NKU), Ministry of Education, Tianjin 300350, China
Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China

***Abstract*—The understanding of password security highly relates to our knowledge of how adversaries guess passwords, and this makes the modeling of guessing attacks a pivotal task. To maximize guessing effectiveness, the adversary generally attempts to guess in descending order of likelihood, akin to the way generative retrieval learning-to-rank works in a recommendation system, which prioritizes information to targeted users based on predicted relevance.**

In this paper, we propose a password guessing framework based on adversarial ranking, named RANKGUESS. We regard the password creation process as sequential decision trajectories. In this context, the adversary is assumed to train an agent where the current state is represented by the password sequence generated up to that point. The action taken is to generate the next token, and the evaluation score assigned by the ranker serves as the reward signal received. Consequently, we frame the problem of password guessing as a Markov Decision Process and tackle it using adversarial ranking techniques. Due to the generality of our framework, RANKGUESS can be applicable to various guessing scenarios (i.e., trawling guessing, targeted password guessing based on personally identifiable information (PII), and conditional password guessing).

By employing 12 large-scale password datasets and six PII datasets, we demonstrate that our models are effective: (1) RANKGUESS surpasses all current state-of-the-art models and outperforms GAN-based methods by 26.29%∼43.69% (avg. 34.80%); (2) When the victim's PII at site A (namely PII_A) is known, RANKGUESS-PII for targeted password guessing based on PII_A, which guesses 58.21%∼91.95% of common users within 10^{12} guesses, outperforms its foremost counterparts by 6.32%∼17.09%; (3) Within 10^7 guesses, our RANKGUESS-MASK based on victims' partial passwords (e.g., d**l*02*), improves the password cracking success rates by 7.70%∼14.85% (avg. 8.21%) compared to its state-of-the-art counterparts. The paper provides a new technical approach to a well-known challenge in the password-guessing field.**

1. Introduction

Passwords are the primary means of user authentication [14], [95] and are crucial for online daily activities of 5.45 billion netizens [8]. With the proliferation of online services, the number of web accounts that users need to manage constantly increases, ranging from 80 to 107 [69]. However, users typically can remember only 5 to 7 different passwords [40]. Unavoidably, users tend to choose popular strings [83], employ personally identifiable information (PII) [84], and are susceptible to password stuffing [7] and tweaking attacks [64]. Recently, users seem to have become accustomed to catastrophic password breaches from high-profile sites (e.g., 3 billion Yahoo leak [2], 1.1 billion Alibaba breach [61]). Even if users' passwords have been stored in slow hash (e.g., PBKDF2) or memory-hard hash (e.g., bcrypt), once these hashes are leaked, users' passwords are subject to offline guessing: the adversary $\mathcal{A}$ can employ dedicated password-cracking hardware like GPU [3] and smart guessing algorithms [86], [87], [91], and a large portion can be recovered/guessed [4], [6]. Thus, it is important to understand how much strength passwords can provide.

To this end, modeling password guessing attacks has become a pivotal undertaking. Guess number [75] has been found to be a good metric to evaluate password strength, and those easily guessed by the adversary $\mathcal{A}$ are considered to of low strength. Besides password hashes, $\mathcal{A}$ may gather extra information (e.g., PII [84], or partial passwords [66]) about the victims to boost her guessing success rates. The incessant escalation of attacks underscores the urgency of investigating the ever-evolving tactics employed by $\mathcal{A}$. Note that, the proportion of users who use password managers is still low in recent years; for instance, BitWarden discovered that only 34% of people use password managers [1]. Even those who use password managers still need a user-chosen master password to protect their password manager vaults.

Overview of password guessing. At ACM CCS'05, Narayanan et al. [60] proposed the first probabilistic guessing model utilizing Markov Chain. Since then, a series of trawling password guessing approaches and methods that employ users' behavior of choosing popular passwords have been proposed, such as PCFG-based [87], Markov-based [54], LSTM-based [56], GAN-based [36], [52], [66], [93], and Random forest-based [86]. As PII is becoming more and more easily gathered, and more often than not, users' PII is leaked together with password hashes [5], [9], it is imperative to characterize PII-aware guessing adversaries. At ESORICS'15, Wang et al. [82] proposed the first PII-aware guessing model Tar-Markov. They worked along this line and proposed a series of PII-aware guessing models like TarGuess-I [84] and RFGuess-PII [86]. Furthermore, there

are cases where $\mathcal{A}$ has somehow obtained (e.g., through side-channel attacks like shoulder surfing [25], keystroke audio feedback [27]) partial passwords (e.g., `d****l*02*`) of the victim. To model this password guessing attack scenario, Pasquini et al. [66] proposed the first conditional password guessing model, CWAE to fill in the unknown characters of the target partial password. At USENIX SEC'23, PassBERT [92] was proposed by using Bidirectional Encoder Representations from Transformers (BERT) with the pre-train and finetuning paradigm to fill the unknown characters in the partial password (e.g., `d****l*02*`).

1.1. Motivation and Challenges

Learning-to-rank (LTR) [28], [59], [77] plays a crucial role in generative retrieval systems [50]: (1) Machine translation: ranking the translations of a set of hypotheses. (2) Computational biology: ranking the candidate 3-D structures in protein structure prediction. (3) Recommendation systems: ranking relevant news articles for users after they have read the current news article or based on user information. Reward models are widely used to evaluate sequence generation in large language modelings (e.g., RLHF [55], using given prompts to get the reward from human evaluation), to align with human preferences as much as possible.

Password guessing further has a clear preference as shown in Equation 1, where the adversary $\mathcal{A}$ usually attempts to guess in descending order of likelihood. Motivated by the above analysis, a natural research question arises: *is it possible to propose a password guessing agent (i.e., guesser), to align with the preferences of learning to rank in password guessing*. How to answer this question is not straightforward, and here we explain why in detail.

In *ad-hoc* password guessing, given an additional *query* information $q \in \mathcal{Q}$ (i.e., PII, and partial password) and a set of N password candidates $\mathcal{G} = \{p_1, p_2, \ldots, p_N\}$ from a password space $\mathcal{C}$, a guessing model f aims to associate a relevance score $f(q, p_n)$ with each pair $(q, p_n) \in \mathcal{Q} \times \mathcal{C}$ to rank the whole password set based on their relevance to q. For example, the ranking model outputs the guessing list $L = [p_N, p_{N-1}, \ldots, p_1]$ if it determines that

$$f(q, p_N) > f(q, p_{N-1}) > \cdots > f(q, p_1), \qquad (1)$$

as $\mathcal{A}$ usually attempts to guess in descending order of likelihood, just as generative retrieval learning to rank. We aim to establish a guessing model capable of interacting with the environment, perceiving the strength of passwords, and thereby generating guesses like the real-world adversary as Figure 1. We divide the aforementioned password guessing attack process into three parts:

- Learning to generate: generate guess candidates for different Scenarios #1-#3 in Table 1 via the guesser.
- Learning to rank: learn which passwords are ranking towards the top in password space $\mathcal{C}$ via the ranker.
- Instructing to guess: with the aforementioned two capabilities, the ranker can instruct the guesser to guess.

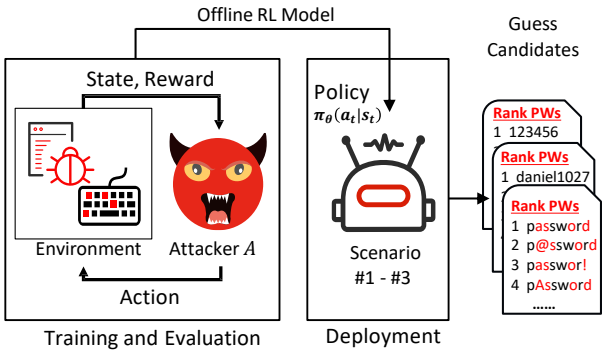


Figure 1: An illustration example of the offline Reinforcement Learning (RL) guessing models for three guessing attacks, i.e, trawling, targeted and conditional password guessing Scenarios #1-#3 as in Table 1. The attacker $\mathcal{A}$ makes decisions about the best next actions to maximize its rewards based on the current state.

Now three challenges emerge:

C1: Even minor errors would result in the failure of password guesses. Passwords are extremely short texts with rich semantics and syntax. Contrastively, small changes in text generation still carry semantics. For example, if a model generates `daniel1027` it would fail at guessing `d@niel1027`. This imposes more constraints on the exact match in Section 1.1 mentioned three tasks [41], [86] and needs more additional restrictions when generating guesses.

C2: How to design rewards for password guessing task. Password guessing can be regarded as a learning-to-rank problem, which generates guesses ranking via likelihood in descending order. Fundamentally, a reward signal is an abstraction of the task and serves as the bridge through which we convey the task objective to the agent. Designing rewards in password guessing can be a daunting task, as defining desired behaviors through reward signals can be challenging or may necessitate numerous expert demonstrations. In the learning process for generating and ranking, we need to assign a suitable reward to each state of the password, followed by the calculation of the cumulative reward collection for the entire password generation trajectory.

C3: How to utilize additional information to guess users' passwords. The construction of passwords typically involves PII or commonly used strings, and the adversary $\mathcal{A}$ may acquire partial passwords (e.g., `d****l*02*`) through various means. How can a guessing model for individual users be learned from the additional available information? Wang et al. proposed solutions for traditional matching methods in TarGuess-I [84] and RFGuess-PII [86]. However, there is no clear-cut solution for deep learning approaches here. The challenge lies in maximizing the success rate by leveraging additional information with a limited guesses.

1.2. Our contributions

In this work, our key contributions are three-fold:

- **A new technical password guessing route.** We build a password guessing model using the adversarial ranking technique, treating passwords as trajectories. We, *for*

TABLE 1: Summary of three password guessing scenarios with toy examples, labeled sets, and corresponding explanations.

Attack scenarios	Toy example ($x \rightarrow y$) with ($x \in X, y \in Y$)	Labeled sets Inputs X, accompanied by its outputs Y	Explanations of attack
Trawling guessing	B_s daniel1027 $\rightarrow$ daniel1027 E_s	Passwords with the begin symbol B_s with them shifted right with the end symbol E_s.	The attacker does not care who the specific target is, and its the only goal is to guess as many passwords as it is possible [81].
Targeted guessing	(daniel@xxx.com,daniel,lavender, 10271985,123-456-7890)$\rightarrow$daniel1027	PII (email, user name, name, birthday, phone, etc.) with corresponding targeted passwords.	Utilizing models to inference password with provided specific users' PII [84].
Conditional guessing	d$\star\star\star\star$l$\star$02$\star$ $\rightarrow$ daniel1027	Partial passwords with complete passwords.	Complete password by filling the pivot [66].

the first time, define the learning to rank rewards for password guessing. Then, We employ ranker-instructed rewards at each timestep t during Monte Carlo rollouts, facilitating policy gradient optimization. The model learns from the leaked password dataset to generate high-quality password guessing candidates.

- **Three improved password guessing models.** We systematically analyze various experimental results. Our RANKGUESS demonstrates exceptional performance than state-of-the-art (SOTA) in trawling guessing scenario by 0.79%$\sim$7.41% (avg. 4.32%) and outperforms GAN-based SOTAs by 26.29%$\sim$43.69% (avg. 34.80%). RANKGUESS-PII, when armed with the victim's personally identifiable information (PII_A) at site A, achieves targeted guessing with a remarkable success rate of 58.21%$\sim$91.95% within 10^{12} guesses, outperforming SOTA by 6.32%$\sim$17.09%. RANKGUESS -MASK, exploits victims' partial passwords (e.g., d$\star\star\star\star$l$\star$02$\star$), and significantly enhances the cracking success rate by 7.70%$\sim$14.85% (avg. 8.21%) within 10^7 guesses, surpassing latest SOTAs.
- **Some insights.** When it comes to predicting the next token in the state *n*-gram patterns, RANKGUESS employs a reward model via learning to rank approach, which converts non-differentiable evaluation function scores to policy gradient updates. RANKGUESS can better detect PII segment in Chinese passwords, and sequential information in English passwords, while enhancing capabilities for capturing substrings (len$\geq$6). We design a password strength meter based on RANKGUESS to enhance user protections. In addition, we evaluate its accuracy and security on real-world passwords. It is capable of resisting both offline and online password guessing attacks in the real world.

2. Background and related work

2.1. Three guessing scenarios

Trawling password guessing. Trawling password guessing means that the adversary $\mathcal{A}$ does not care who the specific target is, and the only goal is to guess more passwords under the guess number allowed (e.g., $> 10^9$ as recommended by Wang et al. [81]). At IEEE S&P'09, Weir et al. [87] proposed the password guessing models based on probabilistic context-free grammar (PCFG) [39]. In 2014, Ma et al. [54] implied that the PCFG algorithm is statistics-based models that crack passwords by counting the frequency of elements in the training set (e.g., the LDS in PCFG), which has the inherent limitations of password sparseness and overfitting [54]. Meanwhile, Narayanan et al. [60] introduced the Markov Chain [38], [70] to capture dependencies between passwords. In 2019, Hitaj et al. [36] proposed PassGAN, a method that learns password samples between the real and generated passwords to simulate password distribution. They use a generative neural network to capture password features and a discriminative neural network [34] to evaluate the similarity between training and generated ones. At this point, it indicates that the generator has the capability to generate real passwords. After that, Pasquini et al. [66] alleviated the mode collapse problem of training GANs and Wasserstein autoencoder via Wasserstein GAN and Context Wasserstein Autoencoder, abbr. as WGAN and CWAE. On this basis, they constructed two password guessing frameworks, that is, Conditional Password Guessing (CPG) and Dynamic Password Guessing (DPG). However, both PassGAN [36] and Pasquini et al. [66] simply utilize ResNet [35] as their generator from random noise and discriminator without accounting for the cyclic dependencies within passwords nature. GENPass [52] is a hybrid model combining PCFG, LSTM, and CNN for password generation and classification. Yu et al. [93] apply the gradient normalization method to improve password training and generation process stability and therefore propose the model GNPassGAN.

Targeted password guessing. There are many kinds of PII about users, such as demographic related personal information (name, birthday, age, phone number, education, gender, etc.). The goal of targeted password guessing is to crack the password of a given targeted user as quickly as possible [84]. In 2015, Wang et al. [82] introduced the first targeted probabilistic model known as Tar-Markov. Grounded in the principles of Markov Chain [60], this approach was designed to leverage the frequency of names within a training password set in order to estimate the likelihood of specific users choosing name-based passwords. In 2016, Li et al. [46] proposed a targeted password guessing model called Personal-PCFG, which builds upon the PCFG [87]. This model utilizes the length of PII to segment and analyze password. After that, Wang et al. [84] showed that their TarGuess-I is more efficient than Personal-PCFG [46] using PII categories to crack passwords, which can gain success rates over 20% with 100 guesses. TarGuess-I adds six more PII tags (name, username, birthday, etc.) to the three basic LDS segments in the PCFG algorithm [39].

Conditional password guessing. Conditional password guessing means that the adversary $\mathcal{A}$ has obtained an incomplete password (e.g., some wildcards) of the victim by exploiting various side channel attacks [25], [27], [66],

[92] (e.g., shoulder surfing, keystroke audio feedback, and thermal residue), then initiates a password guessing attack (e.g., `e***a*rt*`, `d****l*02*`). A plain way to tackle conditional password guessing is by using trawling models to create numerous potential passwords [66]. Then, these potential passwords could be filled into the wildcard through a filtering process. However, the task of efficiently removing irrelevant passwords from this pool can be resource-intensive and demand significant storage capacity.

At IEEE S&P'21, Pasquini et al. [66] proposed the first deep conditional password guessing model based on Wasserstein Autoencoder [73]. They refer to the final model as context wasserstein autoencoder (CWAE) which consists of an encoder embedding a partial password into latent representations, and a decoder converting the latent representations of the partial password into passwords. In 2023, Xu et al. [92] proposed a SOTA approach for conditional password guessing based on bi-directional encoder representation from transformers (BERT) [26] using pre-train and finetuing paradigm [51]. However, PassBERT [92] requires a massive pretraining password to achieve results competitive with CWAE [66], giving it a biased advantage over other models. Additionally, BERT [26] was originally designed to address issues of long-range dependencies and parallel processing between words, which are not pertinent concerns in password guessing. Therefore, choosing Transformer [79] models like BERT family network [26] as the backbone network may not be a suitable option.

3. Setting and RL formulation

3.1. Password guessing modeling

Suppose the password token set in this work is denoted by $\mathcal{V}$, and at timestep t, the previous tokens generated are $\mathrm{PW}_{1:t} = (w_0, w_1, ..., w_{t-1})$ illustrated in Figure 1, where all tokens $w_i \in \mathcal{V}$ are listed in Table 2 (and Table 9 in Appendix A). To formalize this, we regard password generation process $\mathrm{PW}_{1:T} = (w_0, w_1, ..., w_{T-1})$ as a sequential decision process [68] in the form of:

$$\tau = \{s_0, a_0, s_1, a_1, \ldots, s_T, a_T\}. \tag{2}$$

At timestep t, action a_t could be considered as to choose a token from the token set $\mathcal{V}$, with the policy denoted as $\pi_\theta(a_t \mid s_t)$, where the state s_t is the t-gram in previous steps $\mathrm{PW}_{1:(t-1)}$. In such a case, the offline reinforcement learning (offline RL) model aims: to learn an optimal (or nearly optimal) policy from a pre-collected password D without an interactive environment. We use $\mathbb{S}$ and $\mathbb{A}$ to represent the RL models' input and output space formally called state and action in RL scenes as Figure 1. $r_t \in \mathbb{R}$ is the temporal reward for each timestep t, where $\mathbb{R}$ is the real number set. A unit in a pre-collected password called transition is a four-element set: $\{s_t, a_t, r_t, s_{t+1}\}$, where $s_t \in \mathbb{S}, a_t \in \mathbb{A}$, and $s_{t+1} \in \mathbb{S}$ is the successive state of s_t. The probability of a trajectory τ is given by the format of

$$\begin{aligned} P_\theta(\tau) &= P_\theta(s_0, a_0, s_1, a_1, \cdots, s_T, a_T) \\ &= P_\theta(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t) P_\theta(s_{t+1} \mid s_t, a_t). \end{aligned} \tag{3}$$

For example, $\mathrm{PW}_{1:T} = \{\mathtt{P}, \mathtt{a}, \mathtt{s}, \mathtt{s}, \mathtt{w}, \mathtt{@}, \mathtt{r}, \mathtt{d}\}$, we set s_0 as B_s (standing for the begin symbol), then the action a_0 is chosen to obtain the maximum reward with r_0 for the trajectory token $w_0 = \mathtt{P}$, resulting in $s_1 = \mathsf{B}_s\mathtt{P}$, then make next action a_1 to get $s_2 = \mathsf{B}_s\mathtt{Pa}$, and so on, until the complete password $\mathsf{B}_s\mathtt{Passw@rd}\mathsf{E}_s$ is generated. A set of transitions in chronological order forms a trajectory in password D. Based on the transitions, the offline RL model learns the Markov Decision Process (MDP) underneath the passwords and forms a policy $\pi_\theta(a_t|s_t)$ such that the password ranking expected reward $\mathfrak{L}(\theta)$ is maximized [37], [71]:

$$\begin{aligned} \mathfrak{L}(\theta) &= \mathbb{E}_{\tau \sim P_\theta(\tau)}[\mathcal{T}(\tau)] \\ &= \mathbb{E}_{\mathrm{PW}_{1:T} \sim P_\theta(\mathrm{PW}_{1:T})}[\mathcal{T}(\mathrm{PW}_{1:T})], \end{aligned} \tag{4}$$

where $\mathcal{T}(\mathrm{PW}_{1:T})$ represents the total reward for the password $\mathrm{PW}_{1:T}$. This is the first attempt at modeling learning to rank for guessing in this manner, utilizing RL to train a guessing model. Thus, we name our model as RankGuess. The goal is to maximize the reward of the guesser $\mathcal{G}_\theta$.

TABLE 2: The abbreviations and notations in this paper.

Notations	Descriptions
B_s E_s	Begin symbol and End symbol.
PW $\mathrm{PW}_{1:t}$	Password and partial password.
$\mathrm{L_n}$ $\mathrm{D_n}$ $\mathrm{S_n}$	The letters, digits and special segment in length n.
$\mathcal{T}$ $\mathcal{E}$	Reward and expert.
$\mathbf{E}_1 \sim \mathbf{E}_3$	$\mathbf{E}_1$ means username/local part; $\mathbf{E}_2$ means the second-level domain; $\mathbf{E}_3$ means the top-level domain,$\cdots$.
$\mathbf{T}_1 \sim \mathbf{T}_3$	$\mathbf{T}_1$ means the whole phone number; $\mathbf{T}_2$ means three digits/area code; $\mathbf{T}_3$ means subscriber/line number,$\cdots$.
$\mathbf{N}_1 \sim \mathbf{N}_{10}$	$\mathbf{N}$ stands for name usages, $\mathbf{N}_1$ for the usage of full name, $\mathbf{N}_2$ for the abbr. of the full name (e.g., `ds` from `danielsmith`),$\cdots$.
$\mathbf{B}_1 \sim \mathbf{B}_{15}$	$\mathbf{B}$ stands for birthday usages, $\mathbf{B}_1$ for full birthday in MDY format (e.g., `10271985`), $\mathbf{B}_2$ for full birthday in YMD format, $\cdots$.
π (a, s) τ	π means password generation policy; (a, s) means guesser action and state pair; τ means a trajectory of password.

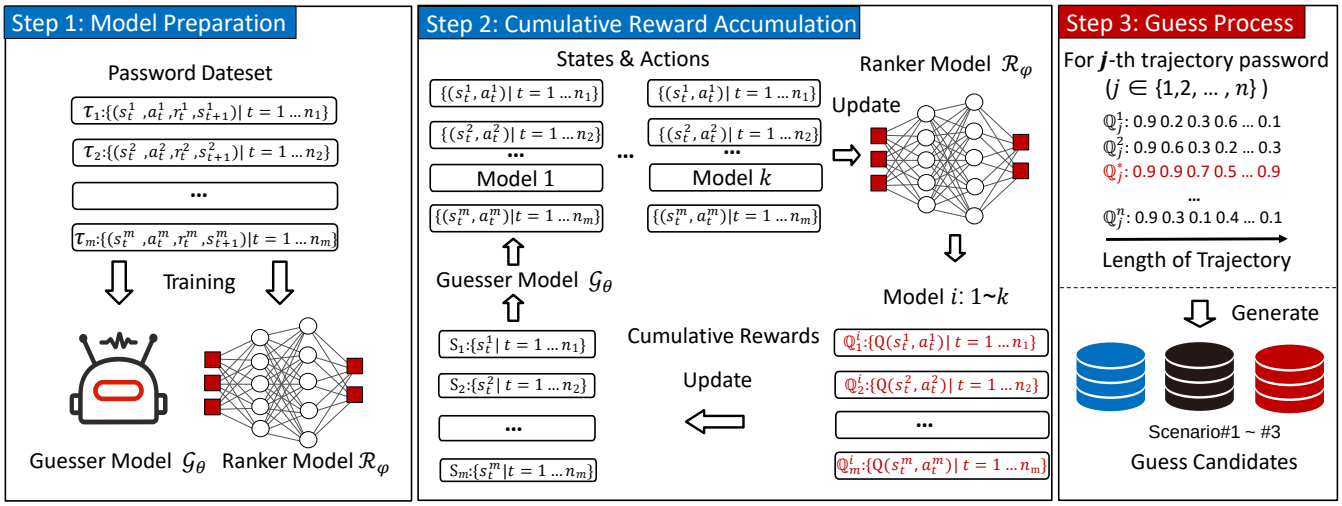


Figure 2: The workflow of RANKGUESS contains three steps, i.e., model preparation, cumulative reward accumulation, and guess process. RANKGUESS firstly pre-train a guesser model and a ranker model via Section 3.3 on the trainset, then collects the cumulative rewards via Section 3.2 from the state-action pairs of password using offline RL model. After k iterations, the training process of the model converges and stabilizes. Finally, RANKGUESS can employ the rewarding process of trajectories and thus be used to generate guesses.

3.2. Learning to rank reward function

Regarding the mentioned reward in Section 3.1, we need to design an appropriate one for password guessing. To determine the strategy our expert employs for guessing, we rank the passwords in the trainset to obtain the top passwords (typically, 1w) or tagged tokens as expert $\mathcal{E}$ to imitate. The task of guessing the password requires a corresponding reward mechanism for password guessing alignment.

Ranking scores. Taking inspiration from the ranking steps used in network search algorithms like PageRank [63], we introduce a similarity metric capable of evaluating relative ranks between PWs and formulate the ranking score of PW concerning a given expert $\mathcal{E}$ and comparison $\mathcal{S}^+$ as follows:

$$P(\mathrm{PW}|\mathcal{E},\mathcal{S}^+) = \frac{\exp\left[\cos(\mathbf{y}_{\mathrm{PW}},\mathbf{y}_{\mathcal{E}})\right]}{\sum_{\mathrm{PW}'\in\mathcal{S}^+\bigcup\{\mathrm{PW}\}}\exp\left[\cos(\mathbf{y}_{\mathrm{PW}'},\mathbf{y}_{\mathcal{E}})\right]}. \tag{5}$$

Here, $\mathbf{y}_{\mathrm{PW}}$ and $\mathbf{y}_{\mathcal{E}}$ are the feature vectors of the input and expert password. For a set, the ranking score is the expectation of each individual, denoted as $\mathcal{R}_\varphi(\mathrm{PW}|\mathcal{E},\mathcal{S}^+)$.

Monte-Carlo (MC) rollouts. Note that in traditional RL, the current reward is influenced by the rewards from intermediate states and future states. However, in password guessing, the guesser $\mathcal{G}_\theta$ obtains the reward if and only if one password has been completely generated, which means no intermediate reward is gained before the sequence hits the end. To relieve this, we utilize the MC rollouts methods [16], [47], [58], [94] to simulate intermediate rewards:

$$\mathcal{Q}_{\theta,\varphi}(\mathrm{PW}_{1:t-1},\mathcal{E}) = \mathop{\mathbb{E}}_{\mathrm{PW}\sim\mathcal{G}_\theta}\left[\mathcal{R}_\varphi(\mathrm{PW}|\mathcal{E},\mathcal{S}^+,\mathrm{PW}_{1:t-1})\right]. \tag{6}$$

Here, PW represents the password sampled by rollout methods at state s_{t-1}. To be more specific, the beginning tokens $(w_0, w_1, ..., w_{t-1})$ are fixed, and the rest tokens are consecutively sampled by $\mathcal{G}_\theta$ until the last token w_T is generated. Assuming we conduct N trials and obtain N trajectories, denoted as $\tau^{(1)}, \tau^{(2)}, \dots, \tau^{(N)}$, with corresponding state value of $\mathcal{Q}_{\theta,\varphi}(\tau^{(i)})$, the state value of s_t can be approximated as:

$$\tilde{Q}_{\theta,\varphi}(\mathrm{PW}_{1:t-1},\mathcal{E}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{Q}_{\theta,\varphi}(\tau^{(n)}). \tag{7}$$

As $N\to\infty$, the empirical state value function estimator $\tilde{Q}_{\theta,\varphi}(\mathrm{PW}_{1:t-1},\mathcal{E})$ converges in probability to the true value $\mathcal{Q}_{\theta,\varphi}(\mathrm{PW}_{1:t-1},\mathcal{E})$ (as demonstrated in Appendix A.2) which we denote as δ_t for short as the empirical state value.

3.3. Workflow

Step. 1: Model Preparation (MP). In the box to the left in Figure 2, the adversary $\mathcal{A}$ prepares the guesser model $\mathcal{G}_\theta$ and the ranker model $\mathcal{R}_\varphi$ based on the target password, which contains m trajectories $\tau^{(i)}$ with the length of T ($i \in \{1,2,\dots,m\}$). The ranker model is optimized to estimate the cumulative reward of each state-action pair. For each trajectory in the password, a series of predictions for its state-action pairs compose the exclusive feature for learning to rank. We use the Monte Carlo Tree Search to optimize the ranker model (see in Section 3.2). In addition, the adversary $\mathcal{A}$ trains a set of guesser models with conventional MLE methods using the supervised pre-training with model initializations. Note that the tokens for different guessing Scenarios #1-#3 are different as in Section 4.

Step. 2: Cumulative Reward Accumulation (CRA). In the middle, the guesser model $\mathcal{G}_\theta$ observes the states of the password and takes actions. For the i-th trajectory in the password, the $\mathcal{R}_\varphi$ observes the state s_t^i and the action a_t^i of each iteration model, where a_t^i represents the guesser model's action at the t-th step of trajectory τ_i, representing the learned policies with model initialization and training

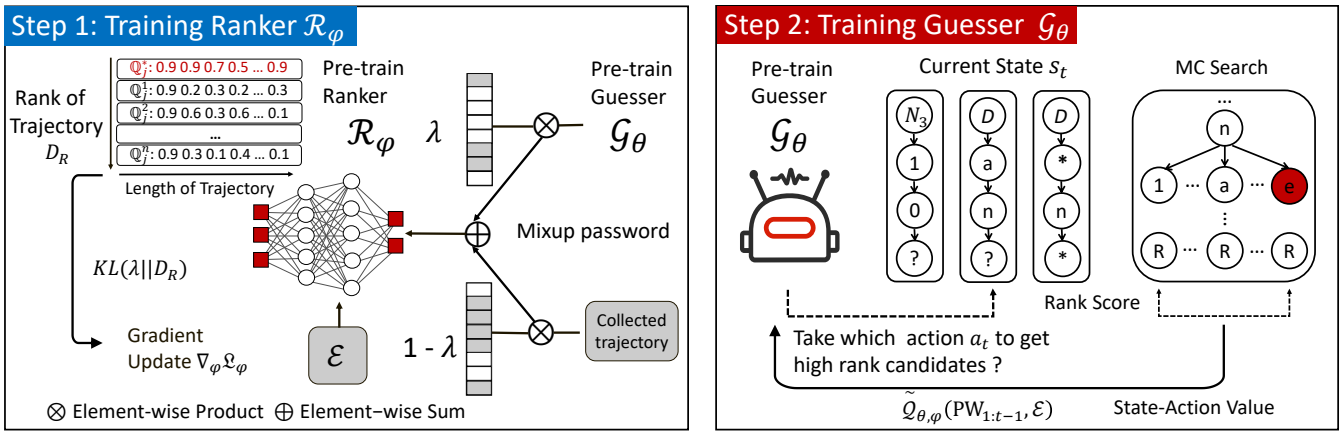


Figure 3: Our RANKGUESS training phases are divided into two steps: (1) training ranker $\mathcal{R}_\varphi$ and (2) training guesser $\mathcal{G}_\theta$. The ranker $\mathcal{R}_\varphi$ is trained to assign corresponding rankings to mixup passwords under the guidance of expert $\mathcal{E}$, which represents the top passwords from the training set (serving as the expert for imitation). $\mathcal{S}^+$ consists of n randomly samples from the training set, $\mathcal{S}^-$ comprises n passwords generated by $\mathcal{G}_\theta$. By mixing these sets with blending ratios ranging from 0 to 1, we create n mixup password sets. The expected distribution is monotonically decreasing due to the changing blending ratios λ. Meanwhile, the training of the guesser $\mathcal{G}_\theta$ generates a password with a high reward passing back to the intermediate value via Monte Carlo Tree Search (MCTS) [17] in Section 3.2. Through these optimizations, the guesser model $\mathcal{G}_\theta$ can generate effective passwords under the guidance of the ranker model $\mathcal{R}_\varphi$.

on the password. Using the learning-to-rank reward in Section 3.2, the ranker $\mathcal{R}_\varphi$ calculates the estimations for all state-action records, i.e., the estimated cumulative rewards, which are the samplings of the exact cumulative rewards. Iteratively, we update the guesser and ranker model k times.

Step. 3: Guess Process (GP). After the above two steps, the guesser $\mathcal{G}_\theta$ obtains the estimated cumulative rewards from the ranker models $\mathcal{R}_\varphi$ and then conducts the guess process. For j-th trajectory, the guesser model $\mathcal{G}_\theta$ knows $\left\{\mathbb{Q}_j^i \mid i \in \{1, 2, \ldots, k\}\right\}$ which obtains the best reward, i.e., $\mathbb{Q}_j^*$. We will detail the algorithms adapted for different password guessing attack scenarios in Section 4.

3.4. Implement detail of workflow

Due to the need for sampling during Step.2, the non-differentiability caused by sampling for discrete token generation makes gradients unable to back-propagate to the guesser $\mathcal{G}_\theta$. Therefore, we need to convert the non-differentiable loss into a reward signal using policy gradient.

Training guesser. With the feasible intermediate rewards, we can finalize the objective function for a complete password. Refer to the proof in Appendix A.3, the gradient of the objective function for guesser $\mathcal{G}_\theta$ can be formulated as:

$$\nabla_\theta \mathfrak{L}_\theta = \mathop{\mathbb{E}}_{\mathrm{PW}_{1:T}\sim\mathcal{G}_\theta}\left[\sum_{t=1}^{T}\sum_{a_t\in\mathbb{A}} \nabla_\theta \delta_t \log \pi_\theta(a_t|\mathrm{PW}_{1:t-1})\right], \tag{8}$$

where ∇_θ is the partial differential operator. $\mathbb{E}_{s_{1:T}\sim\mathcal{G}_\theta}$ is the expectation over all sampled complete password based on the guesser's parameter θ within one batch. Note that we only compute the partial derivatives for θ, as $\mathcal{R}_\varphi$ is fixed during the training of the guesser. Importantly, different from the policy gradients in other works [36], [66], [94], our method replaces the simple binary outputs with a ranking system, which can better reflect the quality of imitate password and facilitate effective training of the guesser $\mathcal{G}_\theta$ (see Figures 12 and 13 in Appendix A.5).

Training ranker. To train the ranker's parameters set φ, we keep the θ parameters fixed. We generate samples $\mathcal{S}^-$ using the guesser $\mathcal{G}_\theta$ and sample the same size passwords from the ground truth as $\mathcal{S}^+ \sim \mathcal{P}_h$ in $\mathcal{D}$. Next, we create mixup shadow samples $\mathcal{S}_i^M = \lambda_i \mathcal{S}^+ + (1-\lambda_i)\mathcal{S}^-$, where blending ratios $\lambda_i \sim \mathcal{U}(N)$. Then, we obtain the mean rank distribution of $\mathcal{T}_{\theta,\varphi}(\mathcal{S}_i^M, \mathcal{E})$, and update ϕ via the equation:

$$\nabla_\varphi \mathfrak{L}_\varphi = -\sum_{i=0}^{T-1} \nabla_\varphi \mathcal{T}_{\theta,\varphi}(\mathcal{S}_i^M, \mathcal{E}) \log \lambda_i, \tag{9}$$

by measuring the difference between the mean rank distribution and the λ_i distribution using Kullback–Leibler divergence. We can find that the two parts of our training have inherent adversarial attributes. That's because the ranker has a razor-sharp sense of what's valid and what's not, while the guesser gains a strong capability to generate guesses:

- The learning objective of the guesser $\mathcal{G}_\theta$ is to have the ranker $\mathcal{R}_\varphi$ assign higher ranking rewards to the generated sample $\mathcal{S}^-$ compared to the real ones $\mathcal{S}^+$.
- The learning objective of the ranker $\mathcal{R}_\varphi$ is to assign lower rewards to the generated sample $\mathcal{S}^-$ via the guesser model $\mathcal{G}_\theta$ compared to the real ones $\mathcal{S}^+$.

In this assumption, the ranker $\mathcal{R}_\varphi$ continuously rewards the guesser $\mathcal{G}_\theta$ under the guidance of expert $\mathcal{E}$ to enhance the the guesser's password generative capability.

Consequently, this can be regarded as a process where $\mathcal{G}_\theta$ and $\mathcal{R}_\varphi$ engage in a minimax game. The proposed password ranker $\mathcal{R}_\varphi$, shares a similar recurrent neural network structure. Firstly, it maps the connected password matrix into embedded feature vectors $\mathbf{y}_s$ through a series of nonlinear functions. Subsequently, ranking rewards for the password

TABLE 3: Basic information of 12 real-world password datasets with PII used in trawling and conditional password guessing experiments, including details on the web service, language, when leaked, unique passwords after filtering, with PII or not, and the size of those datasets.†

Dataset	Web service	Language	When leaked	Total PWs	Length>30	Removed %	Unique PWs	With PII
Taobao	E-commerce	Chinese	Feb., 2016	15,072,418	88	0.01%	11,633,759	
Dodonew	E-commerce	Chinese	Dec., 2011	16,283,140	13,4758	0.15%	10,135,260	
CSDN	Programmer	Chinese	Dec., 2011	6,428,632	0	0.01%	4,037,605	
Wishbone	Social forum	English	Jan., 2020	10,092,037	250	0.01%	5,933,902	
Mate1	Dating website	English	Mar., 2016	27,401,505	12,430	0.06%	11,916,080	
000Webhost	Web hosting	English	Oct., 2015	15,299,907	4,159	0.76%	10,526,769	
Twitter(𝕏)	Social forum	English	Jun., 2016	40,669,963	282,149	0.69%	10,583,709	
LinkedIn	Job hunting	English	Jan., 2012	54,656,615	17,162	0.22%	34,282,741	
Rockyou	Social forum	English	Dec., 2009	32,603,387	3,140	0.07%	14,326,970	
12306	Train ticketing	Chinese	Dec., 2014	129,303	129,303	0	117,808	✓
ClixSense	Paid task platform	English	Sep., 2016	2,222,045	0	0	1,628,018	✓
Rootkit	Hacker forum	English	Feb., 2011	69,330	5	0.01%	56,835	✓

†PWs = passwords, and PII = personally identifiable information. As with [83], [86], we filter out passwords with length >30 or containing non-ASCII.

features $\mathbf{y}_s$ are computed using expert features $\mathbf{y}_{\mathcal{E}}$ extracted in advance by $\mathcal{R}_\varphi$. Secondly, we construct the ranker $\mathcal{R}_\varphi$. Its input is a piece of password, and its output is a value representing the ranking of this password as a genuine password. This output serves as the reward for the input, and it will be passed back to the generator at the end of each round for the update of policy parameters θ in the guesser. Next, we build the guesser $\mathcal{G}_\theta$, incorporating the policy function $\pi_\theta(a_t|s_t)$ mentioned in Section 3.3, where θ corresponds to the parameters of the guesser model $\mathcal{G}_\theta$. The guesser will be updated according to the previously mentioned policy gradient as Equation 8. GRU has two gates: the update gate and the reset gate. To prototype the model discussed, we use GRU [19] as the guesser model $\mathcal{G}_\theta$, which maps the input embedding representations $(\mathbf{e}_0, \ldots, \mathbf{e}_{T-1})$ of the sequence $\mathrm{PW}_{1:T} = (w_0, \ldots, w_{T-1})$ into a sequence of hidden states $(\mathbf{h}_0, \ldots, \mathbf{h}_{T-1})$ by using the update $\mathbf{g}$ recursively,

$$\mathbf{h}_t = \mathbf{g}\left(\mathbf{h}_{t-1}, \mathbf{e}_t\right). \tag{10}$$

Moreover, a softmax output layer $\mathbf{z}$ maps the hidden states into the output action distribution taken by the guesser $\mathcal{G}_\theta$,

$$\mathbf{z}\left(\mathbf{h}_t\right) = \mathrm{softmax}\left(\mathbf{c} + \mathbf{V}\mathbf{h}_t\right), \tag{11}$$

and the $\mathbf{c}$ and $\mathbf{V}$ are bias vectors and weight matrix. To deal with the common vanishing and exploding gradient [33] of the backpropagation through time, we leverage the GRU cells [19] to implement the update $\mathbf{g}$ in Equation 10 to control the sequential trajectory in Algorithm 1.

4. Experimental results

4.1. Trawling password guessing (Scenario #1)

Trawling guessing [86], as one of the most traditional and common attack scenarios, involves attempting password guesses without using any personally identifiable information (PII) or extra information (e.g., partial passwords).

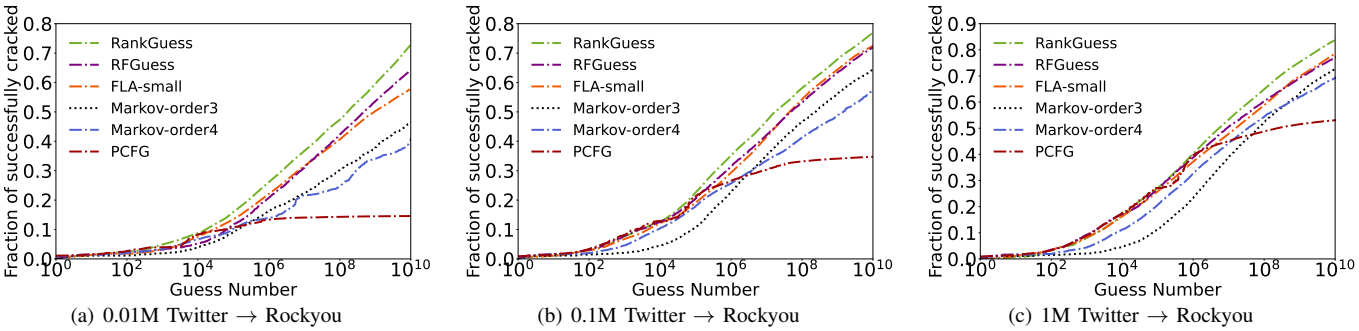
Guesses generation. To align with the length distribution of actual passwords, we have developed an adaptive password generation algorithm. Initially, we start with an empty stack $\mathcal{Q}$ for prefixes and a lookup table LUT for storing printable ASCII characters from 32 to 126. The algorithm iterates by popping a prefix from the stack and predicting token probabilities using guesser $\mathcal{G}_\theta$. If this probability exceeds the threshold or the token is E_s, then the prefix is considered a valid password and is added to the set $\mathcal{P}$.

Algorithm 1: Training RankGuess Framework.

Input: Password or PII tagged set $\mathcal{D}$, training epoch $\mathcal{O}$, iteration count $\mathcal{K}$, state space $\mathbb{S}$, action space $\mathbb{A}$, differentiable policy $\pi_\theta\left(a_t|s_t\right)$, learning rate α, β.
Output: Policy $\pi_\theta\left(a_t \mid s_t\right)$.

Randomly initialize parameters θ and φ;
Pe-train $\mathcal{G}_\theta$ using MLE on $\mathcal{D}$ and update θ.
Generate $\mathcal{S}^-$ samples using $\mathcal{G}_\theta$ for training $\mathcal{R}_\varphi$.
Pre-train D_ϕ via minimizing the cross entropy.
for $t \leftarrow 0$ **to** $\mathcal{O}-1$ **do**
  /* Training ranker model $\mathcal{R}_\varphi$*/
  for $k \leftarrow 0$ **to** $\mathcal{K}-1$ **do**
    Samples $\mathcal{S}^- \sim \mathcal{G}_\theta$ and $\mathcal{S}^+ \sim \mathcal{P}_h$ in $\mathcal{D}$;
    Mixup shadow samples $\mathcal{S}_i^M = \lambda_i \mathcal{S}^+ + (1-\lambda_i)\mathcal{S}^-$, where $\lambda_i \sim U(N)$;
    Get the mean rank distribution $\mathcal{T}_{\theta,\varphi}(\mathcal{S}_i^M, \mathcal{E})$ via Equation. (6);
    $\varphi \leftarrow \varphi - \alpha \sum_{i=0}^{T-1} \nabla_\varphi \mathcal{T}_{\theta,\varphi}(\mathcal{S}_i^M, \mathcal{E}) \log \lambda_i$;
  /* Training guesser model $\mathcal{G}_\theta$*/
  Generate n trajectories $\tau^{(i)} = \left\{s_1^{(i)}, a_1^{(i)}, s_2^{(i)}, a_2^{(i)}, \ldots, s_T^{(i)}, a_T^{(i)}\right\}$ based on the strategy $\pi_\theta\left(a \mid s\right)$;
  for $l \leftarrow 0$ **to** $\mathcal{K}-1$ **do**
    $\delta \leftarrow \mathcal{T}_{\theta,\varphi}(\mathrm{PW}_{1:t-1}, \mathcal{E})$ via MC search;
    $\theta \leftarrow \theta +$
    $\beta \underset{\mathrm{PW}_{1:T}\sim\mathcal{G}_\theta}{\mathbb{E}} \left[\sum_{t=1}^{T} \sum_{a_t \in \mathbb{A}} \nabla_\theta \delta_t \log \pi_\theta(a_t|\mathrm{PW}_{1:t-1})\right]$;
return $\pi_\theta\left(a \mid s\right)$

Otherwise, the prefix is extended with the token, its probability is updated in LUT data structure, and it's pushed back onto the stack for popping. This recursive process continues until the stack is empty. Ultimately, the generation



(a) 0.01M Twitter → Rockyou (b) 0.1M Twitter → Rockyou (c) 1M Twitter → Rockyou

(d) 1M LinkedIn → Taobao (e) 1M Taobao → 000Webhost (f) 1M CSDN → Taobao

Figure 4: Guessing performance of our RANKGUESS in comparison with other SOTA approaches (i.e., RFGuess [86], PCFG [87], 3/4-order Markov [54], [60], and FLA [56]) in the cross-site guessing scenarios with training sets that are of both varied sizes (from 0.01M to 1M) and the fixed size 1M(=1 million), where → means that that the attacker uses the left side data to train the model and uses the right side data as the corresponding test set. Our RANKGUESS performs the best among its counterparts in most testing cases.

algorithm returns the generated guess candidates $\mathcal{P}$.

Algorithm 2: Trawling password generation.

Input: The threshold $\mathcal{T}$ of generated passwords.
Output: Passwords set $\mathcal{P}$.

Init LUT:look-up table{key : $prefix$, value : $prob$}
$\mathcal{Q}.push(\mathsf{B}_s.tokenize())$ /* Implement a tree using a list, build an ordered search tree using BFS.*/
while *!Q.empty()* **do**
  $cur\_prefix \leftarrow \mathcal{Q}.pop()$
  $next\_prob \leftarrow guesser\ \mathcal{G}_\theta(cur\_prefix)$
  $char\_set\ \mathcal{V} \leftarrow$ top_$\omega(cur_prefix, beam_width)$
  for *c in char_set* $\mathcal{V}$ **do**
    $cur\_prob \leftarrow LUT[cur\_prefix] * next\_prob[c]$
    if $cur\_prob > \alpha$ **then**
      if $c == E_s$ **then**
        $\mathcal{P}.append(cur\_prefix)$
      else
        $\mathcal{Q}.push(cur\_prefix + c)$ /* The entire algorithm generates passwords by traversing trajectory τ that meets the criteria. */
        $LUT[cur\_prefix + c] \leftarrow cur\_prob$
return $\mathcal{P}$

Hardware and software dependencies. The experiments are conducted on a workstation operating on Ubuntu 22.04 LTS. The models utilized are implemented using PyTorch and TensorFlow with their corresponding libraries. Notably, the experimental computing machine is equipped with an Intel Xeon Gold 6230R @2.1GHz CPU, 256G Memory, NVIDIA RTX 3090 GPU (with 24GB of VRAM), a 1TB solid-state drive and a 4TB hard disk drive. We posit that this configuration is attainable for real-world adversary.

Datasets. We evaluate the existing password guessing approaches and our RANKGUESS model based on 12 large real-world password datasets (see Table 3), a total of 421.79 million (M) passwords. Eight of our password datasets are from English sites and four are from Chinese ones. Our first step involves removing passwords consisting of symbols beyond the 95 printable ASCII tokens (i.e., the Appendix A in Table 9). We have observed that the original datasets include abnormal passwords that do not seem to be user-chosen passwords or relevant information, either because they are too long (*exceeding* 30 tokens) or too short (*less than* 4 tokens) the same as previous work [56], [66], [86], [87]. Therefore, we have initiated the task of cleaning the datasets before conducting any experiments. These datasets exhibit variations in terms of web service, scale, time, with PII or not, password policy, and language, so it is sufficiently comprehensive to compare the performance of SOTAs.

Evaluation metrics. We compare the performance of different models (limited to 10^{14} guesses). Once the produced guesses hit the targeted password, the account is cracked. We calculate the cracking rate among the account by $\frac{N_{\text{cracked}}}{N_{\text{accounts}}}$.

Ethical considerations. Our datasets are all publicly available on the internet and dark web, and have been widely used in password research [13], [64], [66], [84], [92]. Still, they contain sensitive data (see Table 4), and we take three precautionary measures to ensure that there is no additional harm to users: (1) All our datasets are stored and

processed on computers disconnected from the Internet; (2) Only report the aggregated statistical information and treat each individual account as confidential; (3) Delete all the intermediate sensitive data (e.g., email and datasets of target guesses) once our analysis finishes. While these datasets might be already exploited by the adversary for misconduct, our use is helpful for security administrators/users to measure password strength and prevent weak ones. That is, the defenses (e.g., one can design a password strength meter similar to [15], [56], [76]) derived from our guessing model can be in the public interest. As our datasets are all publicly available from various sources over the Internet, the results in this work are reproducible. Our use of these datasets has been vetted by the security and privacy experts.

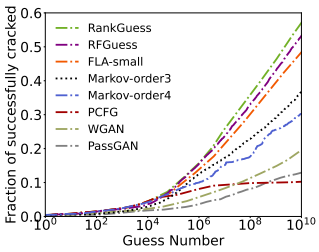


Figure 5: 0.01M Twitter→CSDN

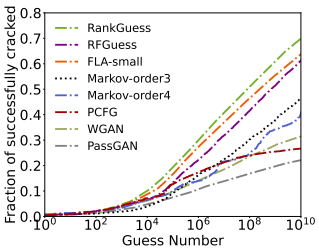


Figure 6: 1M Twitter→Taobao

Experimental setup. We compare RANKGUESS with leading guessing model (i.e., PCFG [87], Markov [54], FLA [56], PassGAN [36] and RFGuess [86]). The setup and hyper-parameters of these compared models are as follows:

- **PCFG**. The PCFG we use is consistent with [54], that is, the probability of L segment comes from the trainset, which surpasses the original Weir et al. [87].
- **Markov**. For Markov, due to the influence of order, this work carries out both 3 and 4-order experiments simultaneously and adopts the Laplace smoothing and end symbol regularization as Ma et al. [54].
- **PassGAN**. The source code (see https://bit.ly/3Ylc F4V) of PassGAN [36] is not optimal as random sampling, so we train it with 10,000 iterations and 10 gradient penalty. We sort the guesses according to their frequency, merging the same passwords.
- **FLA**. We use the source code of FLA [56] (which is available at https://bit.ly/4h2hKXo), and follow its recommended parameters in our experiments. More specifically, we train a model consisting of three LSTM layers with 200 cells (i.e., *small* model in [56]) in each layer and two FC layers, 20 epochs.
- **RFGuess**. The RFGuess parameters we use are consistent with Wang et al. [86], and we train the random forest model using scikit-learn with 30 decision trees. Meanwhile, the minimum of leaf nodes is 10, and the maximum ratio of features is 80%.
- **RANKGUESS**. As detailed in the Algorithm 1, we train RANKGUESS with penalty $\gamma = 0.1$. The loss function is KL divergence, and the optimizer is Adam [42], with an initial learning rate of 0.001 and betas of (0.5, 0.999) for momentum. We train the model with five pre-train epochs and a total of 20 epochs. More specifically, we train the guesser $\mathcal{G}_\theta$ and the ranker $\mathcal{R}_\varphi$ with 17 group numbers and 0.3 dropout rate.

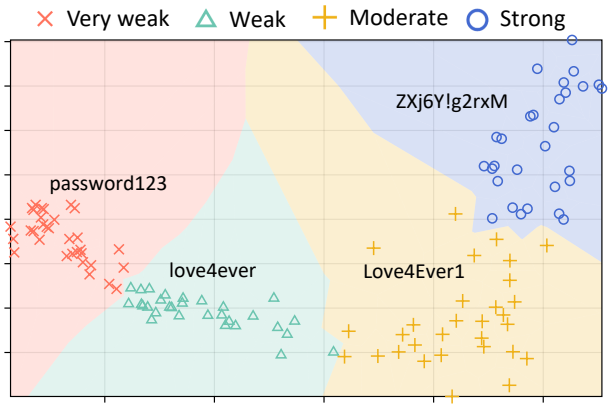


Figure 7:The t-SNE-based visualization of ranker $\mathcal{R}_\theta$ latent space for passwords. We sampled 10,000 passwords, projected them into Ranker's 512-dimensional latent space, then reduced them to 2D using t-SNE [78]. The visualization distinct clusters for very weak, weak, moderate, and strong passwords, and red represents weaker passwords, such as `password123`, followed by green, yellow, and blue. The effectiveness of our Ranker model is consistent with the evaluation of the password strength meters as we expected.

Experimental results. Given the computationally intensive nature of enumerating guesses, we employ the Monte-Carlo algorithm [24], [49] to approximate the number of attempts needed for the adversary $\mathcal{A}$ to discern a password by making guesses in decreasing order of likelihood.

Our experiments are mainly divided into two parts: *(RQ1)* corroborating how much better RANKGUESS performs compared to other SOTA models and *(RQ2)* the impact of various factors on model performance: training size and language in Figure 4, 5 and 6. To answer *RQ1*, we conduct an *apples-to-apples* performance comparison of these approaches using Twitter, LinkedIn, Taobao, and CSDN to crack others (i.e., Figure 4(c)∼4(f), and 6). Initially, the password guessing crack rates of all evaluated models are relatively low, making it difficult to discern any significant advantage. However, the superiority of the RANKGUESS begins to manifest distinctly at 10^4 guesses. In the testing cases, RANKGUESS outperforms the SOTAs by 2.83%∼10.91% (avg. 6.91%) and the GAN-based SOTAs by 26.29%∼43.69% (avg. 34.80%). The distribution differences between different websites are significant and 1M trainset is substantial in differentiating the performance.

Size and Language impact. To answer *RQ2*, we train 0.01M∼0.1M Twitter to crack the same site (i.e., Rockyou). We observed that all models improve with larger training sets. The performance range of PCFG is 14.73% to 53.07%, while RankGuess ranges from 46.63% to 83.55%, particularly excelling with smaller training datasets. Analysis across 1M cases shows varying results; however, categorizing by language reveals distinct patterns. Models perform significantly better on English websites (Figures 4(c) and 4(e)) compared to Chinese sites (Figures 4(d), 4(f) and 6). We analyze the statistics for PW containing a *substring of*

5+ letters (Rockyou 71.69% and 000Webhost 67.82%), and the results are notably higher than that for Chinese (CSDN 28.03%). This discrepancy explains why RANKGUESS performs better on English websites, as the sequence information is stronger. In addition, we visualized the space of the ranker $\mathcal{R}_\varphi$ for randomly sampled 0.1M passwords, reducing the dimensionality from 512 dimensions to 2 dimensions using t-SNE in Figure 7. The `password123` is the weakest, so it appears in the red. Next is `love4ever` in the green category. In the blue category are the strongest passwords (e.g., `ZXj6Y!g2rxM`). It is evident that ranker $\mathcal{R}_\varphi$ exhibits a clear capability learning to rank which instructs the guessing process as mentioned in Section 1.1.

Further analysis. We now reveal the unique advantages of RANKGUESS by comparing the overlap ratios of cracked passwords with RFGuess [86] and FLA-small [56].

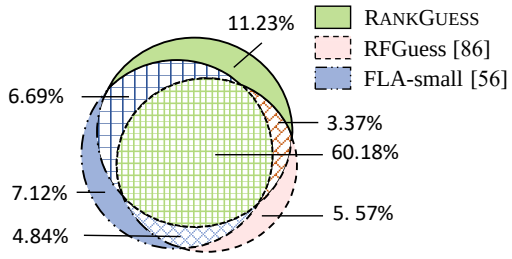


Figure 8: The overlap ratios of cracked passwords.

Overall, 39,254,758 unique passwords are used in our testing sets as conducted in Figure 4, with overlap ratios ranging from 1.36% to 7.60% for English datasets and from 0.54% to 12.81% for Chinese datasets. Figure 8 shows a total overlap of 60.18% (23,623,513 /39,254,758) among the three models. For independently cracked passwords, RANKGUESS has a 11.23% overlap rate (7,325,206 /39,254,758), while that of RFGuess [86] and FLA [56] are 5.57% (3,633,250 /39,254,758) and 7.12% (4,644,298 /39,254,758), respectively. Notably, the intersection of RANKGUESS with FLA [56] is much larger (i.e., 6.69% = 2,626,143 /39,254,758) than the intersection between RFGuess [86] (i.e., 3.37% = 1,322,885 /39,254,758). This highlights that RANKGUESS is closer to the performance of FLA-small [56]. The above analysis reveals RANKGUESS is good at what RFGuess [86] and FLA [56] can do.

To investigate where RANKGUESS has an advantage, we analyze the characteristics of the passwords cracked by different models, focusing on the length and composition of those cracked passwords, as well as providing examples of the cracked passwords in Tables 14 and 15 in Appendix A.6. As illustrated in Figure 16, RANKGUESS outperforms other models by 0.56% in cracking 6-character passwords and by 1.44% in cracking passwords of 11 characters or longer. To measure the similarity between the cracked passwords and the test set, the total variation distance (TVD) between RANKGUESS and the test set is the smallest among all models, indicating that it closely matches the password length distribution of the test set. As shown in Figure 17, RANKGUESS performs well across combinations of mixed letters and digits (LD) and digit-letter (DL) structure.

4.2. Targeted password guessing (Scenario #2)

There are many kinds of PII about users, such as demographic related information (name, birthday, age, gender, etc.). The goal of targeted password guessing is to crack the password of a given user in a given service as quickly as possible [84]. Targeted password guessing predicts the probability of the real passwords (PWs) given PII:

$$P(\text{PW} \mid \text{PII}) = \prod_{\substack{\text{PW}_i \in \text{PWs} \\ p_i \in \text{PII}}} P\left(\text{PW}_i \mid p_i, \mathcal{G}_\theta\right), \qquad (12)$$

where PW denotes the password given personally identifiable information with p_i are six PII attributes.

TABLE 4: Basic information of our PII datasets.

Dataset	Language	Items num	The types of PII attributes
PII-ClixSense	English	2,222,045	Email, UID, Name, Birthday
PII-000Webhost	English	79,580	Email, UID, Name, Birthday
PII-Rootkit	English	69,418	Email, UID, Name, Birthday
PII-12306	Chinese	129,303	Email, UID, Name, Birthday, Phone
PII-CSDN	Chinese	77,216	Email, UID, Name, Birthday, Phone
PII-Dodonew	Chinese	161,517	Email, UID, Name, Birthday, Phone

PII matching algorithm. For passwords containing PII [84], such as `daniel1027` (i.e., $\mathbf{N}_3$: `daniel` in name `danielsmith`, $\mathbf{B}_4$: `1027` in birthday `10271985`, etc.), we have noticed that such passwords can be split into PII segments, such as $\mathbf{N}_3\mathbf{B}_2$. Li et al. [46] introduced an approach for PII matching akin to the PCFG method [39], where, for instance, $\mathbf{N}_3$ denotes exactly three letters in PII, such as `Tom`. It's evident that when PII is already known, splitting the password into segments based on letters as the basic tokens may lead to a loss of entropy of PII [86], which can better preserve semantics as proofed in [86]. For the password `daniel1027`, compared to splitting it into token levels, tagging [10], [48], [53] PII can help avoid the loss of both the name and birthday information. Therefore, we need an effective PII matching algorithm for deep generative models to find PII segments in passwords.

Inspired by Wang et al.'s work [84], [86], the first procedure in our PII matching involves designing a set of tagging rules and establishing a series of tokenized labels to represent personal information. The summarization of these labels is provided in Table 2. Our tagging rules start from 1000 as $\mathbf{N}$, where 1000 represents the complete PII, 1007 denotes the uppercase surname $\mathbf{N}_7$, 1004 represents the lowercase surname $\mathbf{N}_3$, 2000 indicates the complete birthday as $\mathbf{B}$, and so on. We have constructed a comprehensive and full tag-matching representation, illustrated in Figure 14. Based on these rules, a password containing PII can be represented using these labels as the new training trajectory.

The second procedure involves enumerating all possible taggings for each password by substituting PII labels. For instance, consider the targeted password `daniel1027`. It can be represented in different ways: $\{\mathbf{N}_3\,\mathbf{B}_4\}$, $\{\mathbf{N}_3\,1, 0, 2, 7\}$, $\{\mathbf{N}_3\,\mathbf{B}_{10}, 2, 7\}$,$\{\mathbf{N}_3\,1, 0, \mathbf{B}_{11}\}$,$\{\mathbf{N}_3\,\mathbf{B}_{10}\,\mathbf{B}_{11}\}$,$\{\mathbf{E}_3\,\mathbf{B}_4\}\ldots$, and $\{\mathrm{d}, \mathrm{a}, \mathrm{n}, \mathrm{i}, \mathrm{e}, \mathrm{l}, 1, 1, 0, 2, 7\}$, a total of 17 representation methods. In contrast to Wang *et al.*'s works [84], [86], where a single representation is selected as the definitive

TABLE 5: Comparison of four PII-based guessing models.[†]

Experimental setup		RankGuess-PII	RFGuess-PII [86]	TarMarkov 3-order [82]	Tar-Guess-I [84]
Guessing scenario	Guess #				
50% PII-12306 ↓ 50% PII-CSDN	10	15.91%	**16.06%**	15.67%	8.41%
	10^2	**21.37%**	20.40%	20.09%	18.47%
	10^5	28.27%	27.20%	26.35%	**29.01%**
	10^8	**44.90%**	41.00%	43.27%	39.51%
	10^{15}	**70.45%**	66.58%	65.94%	56.05%
50% PII-CSDN ↓ 50% PII-ClixSense	10	4.15%	**5.20%**	4.12%	4.93%
	10^2	6.81%	**7.90%**	7.28%	5.96%
	10^5	**12.40%**	10.96%	11.45%	9.97%
	10^8	**27.83%**	25.46%	23.23%	24.85%
	10^{15}	**94.69%**	90.74%	85.17%	64.51%
50% PII-Dodonew ↓ 50% PII-000Webhost	10	2.51%	**3.07%**	2.40%	2.52%
	10^2	7.33%	**8.00%**	7.10%	6.18%
	10^5	**19.05%**	17.29%	16.50%	14.39%
	10^8	**24.84%**	22.62%	23.45%	21.72%
	10^{15}	**72.22%**	69.61%	68.86%	53.80%
50% PII-ClixSense ↓ 50% PII-12306	10	**14.82%**	11.87%	10.90%	9.12%
	10^2	**20.05%**	18.05%	17.70%	16.67%
	10^5	**32.36%**	30.06%	28.70%	29.15%
	10^8	**57.11%**	54.01%	51.48%	55.75%
	10^{15}	**99.29%**	97.33%	92.38%	78.60%
50% PII-Rootkit ↓ 50% PII-Dodonew	10	**6.96%**	5.95%	6.32%	5.47%
	10^2	**11.40%**	10.07%	10.46%	9.21%
	10^5	**16.17%**	14.88%	15.12%	12.25%
	10^7	**39.45%**	38.81%	27.73%	36.77%
	10^{14}	**89.81%**	86.10%	78.24%	56.91%
50% PII-000Webhost ↓ 50% PII-Rootkit	10	**10.82%**	8.75%	7.90%	6.76%
	10^2	**15.19%**	13.89%	14.10%	12.64%
	10^5	**27.82%**	25.17%	22.32%	21.71%
	10^8	**37.57%**	32.26%	30.34%	31.73%
	10^{15}	**93.93%**	90.45%	85.43%	60.64%

[†] A value in **bold** (attack success rate) indicates that it is the best within each respective row.

one for each password, our generative PII-tagged approaches involve inputting all potential representations into the model during the training process. Our approach deviates from conventional machine learning paradigms, as deep learning necessitates substantial training datasets and places less emphasis on intricate feature engineering and preprocessing techniques. In this context, multiple representations can serve as a part of data augmentation [90]. Our approach can help enhance the robustness and generalization capabilities. We summarize our PII matching in the Algorithm 3.

Datasets. We evaluate the existing approaches and our RankGuess-PII based on six large real-world datasets (see Table 4). As shown in Table 3, three datasets originally contained diverse forms of PII. To facilitate comprehensive targeted password guessing evaluations, we align the passwords that do not initially contain PII with these three PII-containing datasets through email matching. This results in a total of six datasets associated with the PII attribute, detailed in Table 4 along with language and item count.

Approaches for comparison. The prevailing targeted guessing models that utilize personally identifiable information (PII) predominantly consist of TarGuess-I [84], which is rooted in PCFG [87], and TarMarkov [82], which is built upon the Markov model [54]. RFGuess-PII [86], on the other hand, is designed using a random forest classification approach based on *n*-grams. It's worth noting that the original TarMarkov model, as proposed by Wang et al. [82], primarily focuses on leveraging name information. However, it can be readily extended to include additional user details such as user name, birthday, email, and so forth in this paper.

Evaluation metrics. We use pick one of the leaked PII passwords from a specific user as the input, and then infer its variants (limited to 10^{14} guesses). Once the produced variants hit the other password of the same user, the user's account is cracked. We calculate the cracking rate among the evaluation uses/account by $\frac{N_{\text{cracked}}}{N_{\text{accounts}}}$ given different guesses.

Experimental results. In this part, we also propose two research questions: *(RQ3)* How much better is RankGuess-PII compared to other SOTA models? and *(RQ4)* What is the impact of language, size, and PII on the users' password security? To accurately depict the success rates of various attack methods, we conduct 22 testing cases and present specific result values at certain guess numbers (i.e., 10^5 and 10^8 in Table 5). Traditional statistical methods, i.e., TarGuess-I [84], and traditional machine learning methods perform well with small guess numbers (i.e., $\leq 10^3$). However, RankGuess-PII, based on DRL, outperforms than RFGuess-PII [86], TarMarkov-3/4order [84], and TarGuess-I [84] by 6.32% to 17.09% (avg. 12.70%) at 10^5 and 10^8 guesses. It is evident that RankGuess-PII is more stable and better suited for targeted guessing at large guess number.

Algorithm 3: Our PII matching algorithm.

Input: Passwords set $\mathcal{X} = \{pw_1, pw_2, ..., pw_n\}$, mapping from PII to labels $pattern2tags$. /* Pattern2tags are a series of pre-defined (personal information segment tag) tuples, indicating tagging to each PII.*/

Output: Passwords tagging PII structures $\mathcal{P}$.

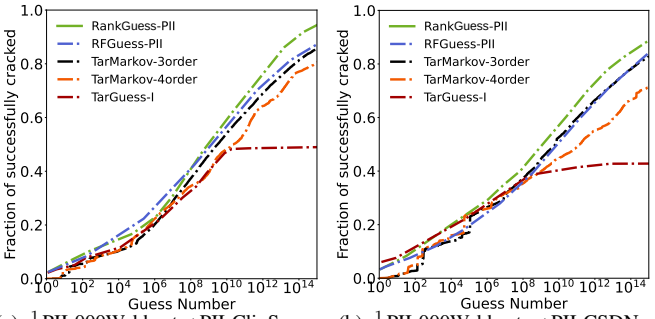
```
for pattern, tag in pattern2tags do
    pos = P.match(pattern);/* Perform string matching to ascertain the presence of PII. */
    if pos! = −1 then
        for i = 0 to pattern.length() do
            for index = i + pos do
                tag_pwd = P.replace(index, tag)
            tag_pwds = PII_match(tag_pwd, pattern2tags);
        /* Conducting DFS, with tag_pwds storing the matched password strings. */
        for i = 0 to pattern.length() do
            tag_pwd = pwd.replace(pos + i, pattern[i]);
for pwd in tag_pwds do
    P.append(unique_tag(tag_pwd));/* Represent each PII with a single tag, eliminating unnecessary tags. */
return P
```

Language, size, and PII impact. To answer *RQ4*, we also perform a *like-for-like* performance comparison of these approaches with regard to dataset size (Exp. #6–#8 in full version) and language. It was also observed that for RankGuess-PII, as the proportion of PII-Mixed training data increased from 50% to 90%, the effectiveness at 10^{14} guesses improved from 91.73% to 99.28%, representing

TABLE 6: Provided are pivots and the coresponding top-5 passwords cracked by RANKGUESS-MASK in Rockyou. The main results are classified according to pivot frequency. For instance, pivot like "**e**aet*" in 10^7 guessing attempts yield a success rate of 95.20%, in which "**elezabeth**", a well-known female first name, emerges as the top-5 probable password. The blank spaces indicate that the passwords were not successfully guessed. The Crack(%) row indicates the crack rate for example pivots.

Pivot	***Common***			***Uncommon***			***Rare***			***Super-rare***		
Rank	****nnie*****	***ar*an*****	***u****123**	**l*ng***n***	***i**e*bo***	**e***a*et***	**1*3**67**0**	**my**ss**r***	**sli*****w**	***a*r*l*pp*r**	****o*eso*c*r**	**1*51*1*86****
1	minniewow	darkangel	august123	longhuong	fisherboy	elizabeth	1234567890	mypassword	sliveshow	hairclipper	ilovesoccor	10519148650
2	hunniebee	farhana11	bunnie123	longhorn1	singerboy	ericareta	1234567880	mypassw0rd	sliv0show		ilovesoccer	13512108611
3	hannie123	harkangel	hungry123	longhorns	silverboy	elixabeth	1234567650	mylesstorm	slideshow		llovesoccer	13510138639
4	jonniecel	maryangel	butter123	langhorn!	mickeyboy	elysabeth	1334567980	mypasswor1	slidwshow			
5	minnie123	marian123	turtle123	longhorn7	cinderbob	**elezabeth**	1234567000	mypasswor8	slinkylaw			
Crack (%)	**99.13**	**78.26**	**86.91**	**69.48**	**83.72**	**95.20**	**72.67**	**91.19**	**85.09**	**28.53**	**43.71**	**56.76**

an increase of 7.55%. We observe that the crack rate for Chinese is, avg. 9.83% higher than that for English, as shown in Table 13 #3A∼#3B and Table 5. In analyzing password patterns, it was observed that the occurrence of `Date_YYMMDD` and `11-digit phone numbers` in Chinese password sets ranged from 16.67%∼27.30%, with an average of 20.23%. This is significantly higher compared to the 2.01% average for English passwords. This indicates that Chinese users' passwords include a higher proportion of PII, making them more susceptible. This is also a key reason for the improved effectiveness of RANKGUESS-PII, as it more effectively identifies PII within users' passwords.



(a) $\frac{1}{2}$PII-000Webhost→PII-ClixSense (b) $\frac{1}{2}$PII-000Webhost →PII-CSDN

Figure 9: Guessing performance of our RANKGUESS in comparison with other targeted approaches using PII (i.e., Targuess-I [84], 3/4-order TarMarkov [82], RFGuess-PII [86]) in the cross-site targeted scenarios. Our RANKGUESS-PII demonstrates superior performance compared to counterparts, especially at large guess numbers, where → means the adversary $\mathcal{A}$ uses the left dataset to attack the right and fractions represent proportions for training.

4.3. Conditional password guessing (Scenario #3)

The adversary $\mathcal{A}$ has obtained an incomplete password of the victim by exploiting various side-channel attacks [66], [92] (shoulder surfing [25], keystroke audio feedback [27], etc.), then initiates a conditional guessing attack. Conditional guessing models crack passwords given a partial one (e.g., `e⋆⋆⋆a⋆et⋆`, `d⋆⋆⋆⋆l⋆02⋆`). A simple way [66] to tackle this issue is by using trawling models to create numerous potential passwords. However, the task of efficiently removing irrelevant passwords from this pool can be resource-intensive and demand significant storage [92]. In this work, the symbol $\star$ serves as a wildcard character. For example, the partial passwords (denoted as pivot) `e⋆⋆⋆a⋆rt⋆` signifies a list of passwords with `elizabeth`.

Conditional password generation. Conditional password guessing [66], [92] predicts the probability of the real passwords (PWs) conditioned on the pivot:

$$P(\text{PW} \mid pivot) = \prod_{\substack{\text{PW}_i \in \text{PWs} \\ p_i \in pivot}} P\left(\text{PW}_i \mid p_i, pivot, \mathcal{G}_\theta\right). \quad (13)$$

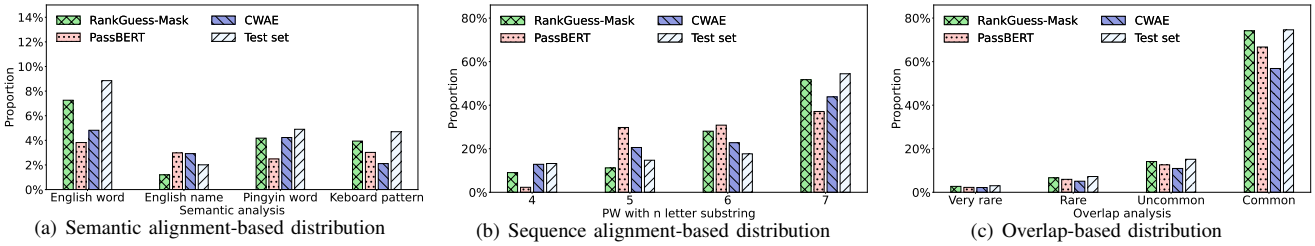
Finding an optimal guessing strategy that minimizes the expected guess rank has a high time complexity. However, the adversary $\mathcal{A}$ could still find approximate solutions that minimize the expected guess rank. We design a Breadth-first search (BFS) with pruning as Algrithm 4. It generates a set of guesses $\mathcal{P}$, given pre-collected partial passwords (denoted as pivot), the number of guesses n, and the number of masked positions m. It employs an iterative process to investigate potential characters for each wildcard position.

Datasets. We evaluate the existing approaches and our RANKGUESS-MASK based on eight large real-world password datasets (see Table 3) and their combinations. Five of our password datasets are from English sites and three are from Chinese ones. In strict adherence to the evaluation meticulously outlined in previous cutting-edge research [66], [92], we systematically apply randomly masked characters with a precisely calibrated 50% within a password. Furthermore, as a stringent criterion, we meticulously retain only those ingeniously generated pivots containing a minimum of five wildcards and four discernible as [66], [92].

Evaluation. Our evaluation aligns with both RANKGUESS-MASK and existing methods like CWAE [66] and PassBERT [92]. We generated a substantial pool of 10^7 candidate passwords for each evaluation pivot. We classified these evaluation pivots into four categories based on the number of passwords satisfying each pivot, denoted as N_{pivots}: (1) ***common*** if $N_{\text{pivots}} \in [1000, 1500]$; (2) ***uncommon*** if $N_{\text{pivots}} \in [50, 150]$; (3) ***rare*** if $N_{\text{pivots}} \in [10, 15]$; (4) ***super-rare*** if $N_{\text{pivots}} \in [1, 5]$. These categories provide a nuanced measurement of pivot guessing performance across different pivot frequencies. Specific illustrative examples of evaluation pivots can be found in Table 6.

Evaluation metrics. The evaluation metrics are the average cracking rates among evaluation pivots in each pivot class. The cracking rate for each pivot is $\frac{N_{\text{intersection}}}{N_{\text{pivots}}}$, where the $N_{\text{intersection}}$ is the intersection between the 10^7 candidates and the passwords satisfying the pivot in an evaluation set.

Experimental settings. We compare RANKGUESS-MASK



(a) Semantic alignment-based distribution (b) Sequence alignment-based distribution (c) Overlap-based distribution

Figure 10: Semantic and syntactic metric distributions of independently cracked passwords by three major models. Our comparison is structured along three dimensions: semantic information, sequence information, and overlap-based analysis as the same as Figure 8. We systematically examine the advantages RankGuess-Mask gained across different models and test sets in all cases from #A1 to #C2 in Table 7. RankGuess-Mask excels at cracking passwords that consist of English and Pinyin words, as well as keyboard patterns and substrings longer than 6 letters (length≥6). In addition, it also achieves the best crack rate for *common* pivots in Table 10.

Algorithm 4: Pruned BFS for conditional attack.

Input: Password pivot p, guess number n, masked number m.
Output: Passwords set $\mathcal{P}$.

Search width $\omega = \lceil \sqrt[m]{n} \rceil$; /* Calculate the possible states for each position p. */
Empty queue $\mathcal{Q}$ with parallelizable pop operations;
Initialize the queue $\mathcal{Q}$, and $\mathcal{Q}.push((initial\_state, prob = 1.0))$;
for masked position p_1 to p_n **do**
  while $!\mathcal{Q}.empty()$ **do**
    $cur\_tuple = \mathcal{Q}.parallel\_pop()$;
    for *cur_pivot, cur_pivot_prob* **in** *cur_tuple* **do**
      $next\_probs = guesser\ \mathcal{G}_\theta(cur\_pivots)$;
      Select top-ω $next\_probs$ and $chars$ /* Select ω candidate characters for each p_i in the pivots template. */
      for *c,next_prob* **in** *chars,next_probs* **do**
        $next\_pivot = cur\_pivot.replace(p_i, c)$
        $new\_prob = cur\_pivot\_prob * next\_prob$
        $\mathcal{Q}.push(next\_pivot, new\_prob)$ /* Search for possible characters at each position p_i in the pivots. */
Ouput password candidates $\mathcal{P} = \mathcal{Q}.parallel\_pop()$
return $\mathcal{P}$

with two models, i.e., CWAE [66] and PassBERT [92]. The setups of RankGuess-Mask are identical to those outlined in Section 3.3, while the others are as follows:

- **CWAE.** The CWAE employed is consistent with Pasquini et al. [66] using the code (see https://github.com/pasquini-dario/PLR) with the default parameters.
- **PassBERT.** For PassBERT [92], we use the source code (see https://github.com/snow0011/PassBertStrengthMeter) of Xu et al. [92], and follow its recommended parameters in our experiments. More specifically, to fine-tune on the pre-trained model released by PassBERT [92] (i.e., pre-training on 60 M passwords from `Rockyou-2021`), we train a model consisting of two hidden layers with every 128 hidden sizes, gelu hidden activation and Dropout rate of 0.1.

TABLE 7: Comparison of conditional guessing models.†

Experimental setup		RankGuess-Mask	Pass-BERT [92]	CWAE [66]
Guessing scenario	Pivot			
CSDN → LinkedIn	*common*	85.57%	82.38%	95.44%
	uncommon	76.41%	72.86%	83.29%
	rare	66.50%	68.19%	84.84%
	super-rare	74.96%	60.29%	73.19%
CSDN → 000Webhost	*common*	79.14%	75.67%	76.87%
	uncommon	78.57%	74.26%	71.48%
	rare	40.02%	39.69%	38.94%
	super-rare	18.30%	24.81%	15.02%
#A1: Engmix → Rockyou	*common*	99.41%	89.42%	76.20%
	uncommon	93.09%	83.10%	72.19%
	rare	92.30%	82.21%	70.35%
	super-rare	90.29%	74.81%	72.74%
#A2: Engmix → Taobao	*common*	89.14%	85.53%	88.15%
	uncommon	73.09%	70.26%	71.95%
	rare	75.86%	69.58%	71.90%
	super-rare	73.09%	70.59%	68.42%
#B1: Chmix → Rockyou	*common*	92.12%	91.80%	89.13%
	uncommon	89.52%	86.72%	83.19%
	rare	85.17%	81.16%	80.57%
	super-rare	83.46%	74.72%	50.19%
#B2: Chmix → Taobao	*common*	98.10%	96.72%	95.53%
	uncommon	93.47%	91.36%	90.65%
	rare	89.13%	87.30%	85.41%
	super-rare	84.19%	81.23%	80.58%
#C1: Mixed → LinkedIn	*common*	99.81%	97.29%	95.39%
	uncommon	94.01%	91.22%	90.94%
	rare	91.72%	87.32%	90.37%
	super-rare	89.85%	86.37%	85.68%
#C2: Mixed → Dodonew	*common*	98.59%	97.34%	94.71%
	uncommon	97.21%	95.99%	91.31%
	rare	91.35%	90.92%	88.87%
	super-rare	80.35%	60.41%	72.32%

† A value with dark gray (resp. light gray) represents the highest one (resp. 2nd one).

Experimental results. To comprehensively evaluate our model's guessing performance and investigate its generalization ability (i.e., cross-site password guessing scenarios), we design 14 password guessing attack cases (see the first column of Table 7 as well as Table 10) in Appendix 3.1. We divide the password sets into Chinese (CSDN, Dodonew, Taobao), English (Wishbone, 000Webhost, LinkedIn), mixed Chinese (i.e., mix all Chinese passwords), mixed English password sets (mix all English passwords), and all mixed. Notably, in the Rockyou → Clixsense case at 10^7 guesses, it

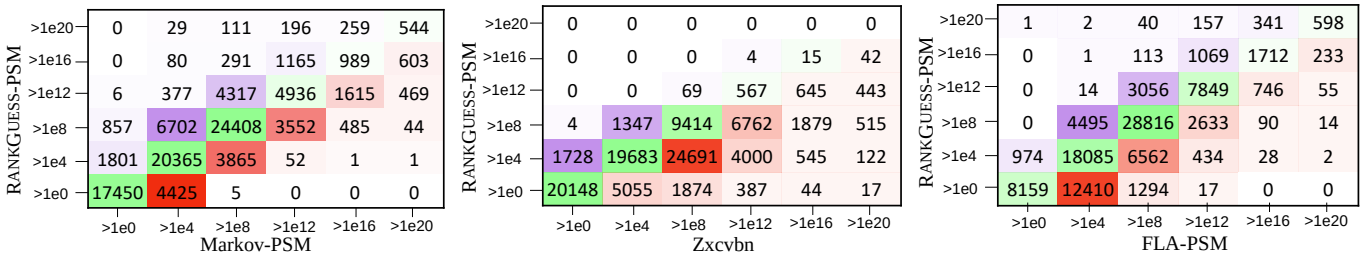


Figure 11: Unsafe errors in PSMs. The utmost significance of this lies in the initial row, as it pertains to the quantity of passwords that are grossly overestimated by other meters. Our RankGuess meter evaluates a total of 7,377 passwords as being guessable within the range of 1 to 10^4 guessing attempts, while the Zxcvbn [88] meter categorizes them as requiring more than 10^4 attempts. RankGuess-PSM and FLA-PSM [56] show similar performance on unsafe errors. The exaggerations of strength are visually and graphically represented in various shades of bold red, while underestimations are depicted in deep shades of purple, and precise evaluations are denoted in soothing shades of green. The color intensity escalates significantly by the number of passwords within a specific category, as [32], [56], [91].

outperforms better than CWAE [66] by 14.85%, relatively. Across all the cases, RankGuess-Mask shows an average success rate improvement of 8.21%, providing significant evidence of its superior cracking performance compared to PassBERT [92] and CWAE [66]. To evaluate the model's guessing performance at various guessing levels: low, moderate, and high guess number (i.e., 10, 10^4, 10^7). Based on the experimental results in Table 10, our RankGuess-Mask outperforms the SOTAs level.

Language and PII impact. Moreover, we study the model's cracking performance when cracking passwords across languages (i.e., Chinese → English, and vice versa) to investigate its generalization. In cases involving Chinese, English, and mixed training sets, RankGuess-Mask performs the best in almost all cases, improving by 7.09%. This shows the generalization ability of RankGuess-Mask. It is observed that RankGuess-Mask does not exhibit a clear preference for specific English or Chinese languages or PII.

Case studies. A significant research question *(RQ5)* is identifying which types of passwords that RankGuess-Mask excels at cracking. To concretely answer it, we analyze passwords generated by different models against our pre-collected semantic dictionaries (detailed in the full version English word, Name, Pinyin word, and Keyboard pattern). We discovered that RankGuess-Mask performs well in generating English, Pinyin words, and keyboard patterns, but is less adept with names compared to PassBERT [92]. RankGuess-Mask showed improved results for passwords with more than five characters, aligning better with the ground truth distribution. Our findings also indicate that passwords with common pivots were successfully cracked, demonstrating the method's effectiveness in distinguishing pivot categories, which highlights the guessing capability of RankGuess-Mask in the conditional scenarios.

5. Implications and insights

As one of the important applications of password cracking algorithms, some password strength meters (PSMs) aim to accurately measure password strength from the adversary's perspective. PSMs [15], [22], [23], [29], [31], [32], [65], [74], [75], [81] show the strength of passwords to help users change a weak password to a strong one, since users' perception of password security is generally unreliable [81]. Labeling passwords as weak may cause usability problems while classifying them as strong could introduce security issues. Additionally, we have implemented a PSM using RankGuess. Our PSM can use a guessing algorithm to calculate the number of guesses required to guess a password and can output the reward for each character, thus prompting the user to make modifications. For specific implementations and examples, please refer to Appendix A.5. To evaluate the effectiveness of our meter, we evaluate its performance using two accuracy metrics [31], [81] to detect unsafe errors [56].

On the accuracy of PSMs. To obtain general results across meters in the real world, we refer to practices [31], [81] that count how meters' guesses for a given password are correlated with occurrence frequencies in an evaluation set using the Spearman coefficient (r_s) [89] as Golla et al. [31]:

$$r_s = 1 - \frac{6\sum d_i^2}{n\,(n^2-1)} \tag{14}$$

where $d_i = \mathrm{rank}\,(\mathtt{PSM}) - \mathrm{rank}\,(\mathtt{FRE})$, represents the difference between the password strength ranks in PSMs and real frequency for each password i in the test set, and n is the number of passwords considered in the calculation.

TABLE 8: Accuracy(r_s) across PSMs, determined through the Spearman correlation of five models. **Bold** value signifies the best.

Dataset	***English***		***Chinese***	
PSMs	LinkedIn	Rockyou	CSDN	Dodonew
RankGuess	**0.652**	0.511	**0.728**	**0.535**
FLA-PSM [56]	0.367	0.355	0.643	0.322
Zxcvbn [88]	0.579	0.463	0.514	0.475
Markov-PSM [15]	0.201	0.337	0.268	0.297
Min_auto [76]	0.558	**0.694**	0.575	0.423

As shown in Table 8, our meter achieves better accuracy than the Zxcvbn meter, and in half of the cases, it is better than Min_auto [76]. In evaluation sets using CSDN 0.1M, RankGuess-PSM has a correlation of 0.728, compared with 0.643 and 0.514 in FLA-PSM [56] and Zxcvbn [88]. Among them, Markov-PSM performs the worst. Our RankGuess-PSM provides a more precise measurement of password strength according to the results.

Unsafe errors. The unsafe errors [56] refer to rating easy-to-guess passwords as strong ones or namely, overestimation of password strength. We investigate these meters' unsafe errors and show the average results using 100,000 passwords randomly sampled from CSDN. Our RANKGUESS-based PSM demonstrated superior accuracy, with up to 1.7 times less unsafe errors compared to Markov-PSM [15] and FLA-PSM [56] as in Figure 11. We use our tuned RANKGUESS-PSM described in Section 3.3. In addition, we find that, compared to others, the Zxcvbn meter's errors are often at very low guess numbers, which can be particularly unsafe. Overestimation of password strength can be disastrous since the seemingly secure passwords could be actually not safe or very vulnerable. For example, Zxcvbn [88] makes 183 utmostly unsafe errors. *Overall, our* RANKGUESS*-based PSM has relatively few unsafe errors but is more accurate than other PSMs as in the Figure 15 and Table 8.*

6. Conclusion

In this paper, we introduce RANKGUESS, a novel password guessing framework that employs the learning-to-rank approach within a Markov Decision Process. This model characterizes password generation as a sequential decision-making process, realizing the goal of password guessing as learning to rank. Tested across three real-world guessing scenarios, RANKGUESS significantly outperforms existing state of the art models, in 29 out of 32 major password guessing cases. Further, we propose a password strength meter to mitigate the risks of real-world password guessing attacks. This work provides a brand new technical route for modeling human-chosen password guessability.

Acknowledgement

The authors are grateful to the shepherd and anonymous reviewers for their invaluable comments on improving the paper. Ding Wang is the corresponding author. This research was in part supported by the National Natural Science Foundation of China under Grants Nos. 62172240 and 62222208, and by the Fundamental Research Funds for the Central Universities, Nankai University (Grant No. 63243154). See the full version of this paper at https://bit.ly/3zIakaR .

References

[1] *What Keeps People From Using Password Managers?*, https://www.wsj.com/articles/what-keeps-people-from-using-password-managers-11623086700.

[2] *Yahoo's 2013 Email Hack Compromised Three Billion Accounts*, https://www.wired.com/story/yahoo-breach-three-billion-accounts/.

[3] *Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331"*, 2013, http://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your-passwords/.

[4] *Bcrypt password cracking extremely slow?* , 2020, https://www.cqure.nl/nl/kennisplatform/bcrypt-password-cracking-extremely-slow-not-if-you-are-using-hundreds-of-fpgas.

[5] *Hacker shares 40 million Wishbone user records for free*, 2020, https://www.bleepingcomputer.com/news/security/hacker-shares-40-million-wishbone-user-records-for-free/.

[6] *11 million+ Ashley Madison passwords already cracked*, 2024, http://arstechnica.com/security/2015/09/once-seen-as-bulletproof-11-million-ashley-madison-passwords-already-cracked/.

[7] *1Password Security Design*, 2024, https://1passwordstatic.com/files/security/1password-white-paper.pdf.

[8] *Number of internet and social media users worldwide as of July 2024*, 2024, https://www.statista.com/statistics/617136/digital-population-worldwide/.

[9] *What happened in the Neopets data breach?*, 2024, https://www.twingate.com/blog/tips/neopets-data-breach.

[10] A. Amini, T. Liu, and R. Cotterell, "Hexatagging: Projective dependency parsing as tagging," in *Proc. ACL 2023*, pp. 1453–1464.

[11] Bitwarden, *Bitwarden Security Whitepaper*, 2024, https://bitwarden.com/help/bitwarden-security-white-paper/.

[12] J. Blocki, B. Harsha, and S. Zhou, "On the economics of offline password cracking," in *Proc. IEEE S&P 2018*, pp. 853–871.

[13] J. Bonneau, "Guessing human-chosen secrets," Ph.D. dissertation, University of Cambridge, UK, 2012.

[14] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The request to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE S&P 2012*.

[15] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive password-strength meters from Markov models," in *Proc. NDSS 2012*.

[16] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, "An adaptive sampling algorithm for solving markov decision processes," *Operations Research*, vol. 53, pp. 126–139, 2005.

[17] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai," in *Proc. AAAI 2008*.

[18] Z. Chen and X. Zhang, "A reinforcement learning-based sequence generation algorithm for password guessing," in *Proc. GLOBECOM 2022*, pp. 4891–4896.

[19] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proc. EMNLP 2014*.

[20] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Proc. NeurIPS 2017*, pp. 1–9.

[21] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Proc. NeurIPS 2017*, 2017, pp. 4299–4307.

[22] X. de Carné de Carnavalet and M. Mannan, "From very weak to very strong: Analyzing password-strength meters," in *Proc. NDSS 2014*.

[23] X. de Carné de Carnavalet and M. Mannan, "A large-scale evaluation of high-impact password strength meters," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 1, pp. 1–32, 2015.

[24] M. Dell'Amico and M. Filippone, "Monte carlo strength evaluation: Fast and reliable password checking," in *Proc. ACM CCS 2015*.

[25] G. Dessouky, T. Frassetto, and A. Sadeghi, "Hybcache: Hybrid side-channel-resilient caches for trusted execution environments," in *Proc. USENIX SEC 2020*, pp. 451–468.

[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT 2019*, pp. 4171–4186.

[27] X. Dong, Z. Shen, J. Criswell, A. L. Cox, and S. Dwarkadas, "Shielding software from privileged side-channel attacks," in *Proc. USENIX SEC 2018*, pp. 1441–1458.

[28] K. Duh and K. Kirchhoff, "Learning to rank with partially-labeled data," in *Proc. SIGIR 2008*, pp. 251–258.

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

[30] L. Gao, J. Schulman, and J. Hilton, "Scaling laws for reward model overoptimization," in *Proc. ICML 2023*, 2023, pp. 10 835–10 866.

[31] M. Golla and M. Dürmuth, "On the accuracy of password strength meters," in *Proc. ACM CCS 2018*, pp. 1567–1582.

[32] M. Golla, M. Wei, J. Hainline, L. Filipe, M. Dürmuth, E. Redmiles, and B. Ur, ""what was that site doing with my facebook password?" designing password-reuse notifications," in *Proc. ACM CCS 2018*.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[34] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR 2016*, pp. 770–778.

[36] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," in *Proc. ACNS 2019*.

[37] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proc. NeurIPS 2016*, pp. 4565–4573.

[38] J. Justesen and T. Høholdt, "Maxentropic markov chains," *IEEE Trans. Inf. Theory*, pp. 665–667, 1984.

[39] J. Justesen and K. J. Larsen, "On probabilistic context-free grammars that achieve capacity," *Inf. Control.*, pp. 268–285, 1975.

[40] M. J. Keith, B. B. M. Shao, and P. J. Steinbart, "The usability of passphrases for authentication: An empirical field study," *Int. J. Hum. Comput. Stud.*, pp. 17–28, 2007.

[41] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, and etc., "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *Proc. IEEE S&P 2012*, pp. 523–537.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR 2015*, pp. 1–15.

[43] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," in *Proc. NeurIPS 2023*.

[44] LastPass, *LastPass Technical Whitepaper*, Aug. 2024, https://assets.cdngetgo.com/da/ce/d211c1074dea84e06cad6f2c8b8e/lastpass-technical-whitepaper.pdf.

[45] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Proc. NeurIPS 2014*.

[46] Y. Li, H. Wang, and K. Sun, "A study of personal information in human-chosen passwords and its security implications," in *Proc. IEEE INFOCOM 2016*, pp. 1–9.

[47] K. Lin, D. Li, X. He, M. Sun, and Z. Zhang, "Adversarial ranking for language generation," in *Proc. NeurIPS 2017*, pp. 3155–3165.

[48] M. Lindemann, A. Koller, and I. Titov, "Compositional generalization without trees using multiset tagging and latent permutations," in *Proc. ACL 2023*, pp. 14 488–14 506.

[49] P. Liu, J. Blocki, and W. Bai, "Confident monte carlo: Rigorous analysis of guessing curves for probabilistic password models," in *Proc. IEEE S&P 2023*, pp. 626–644.

[50] T.-Y. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, pp. 225–331, 2009.

[51] X. Liu, K. Ji, Y. Fu, W. Tam, Z. Du, Z. Yang, and J. Tang, "P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks," in *Proc. ACL 2022*, pp. 61–68.

[52] Y. Liu, Z. Xia, P. Yi, Y. Yao, T. Xie, W. Wang, and T. Zhu, "Genpass: A general deep learning model for password guessing with pcfg rules and adversarial generation," in *Proc. ICC 2018*, pp. 1–6.

[53] L. Loukas, M. Fergadiotis, I. Chalkidis, E. Spyropoulou, P. Malakasiotis, I. Androutsopoulos, and G. Paliouras, "Finer: Financial numeric entity recognition for XBRL tagging," in *Proc. ACL 2022*, pp. 4419–4431.

[54] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE S&P 2014*, pp. 689–704.

[55] P. Mahmoudieh, D. Pathak, and T. Darrell, "Zero-shot reward specification via grounded natural language," in *Proc. ICML 2022*.

[56] W. Melicher, B. Ur, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean and accurate: Modeling password guessability using neural networks," in *Proc. USENIX SEC 2016*, pp. 175–191.

[57] P. Memberships, *1Password Security Design*, Oct. 2024, https://1passwordstatic.com/files/security/1password-white-paper.pdf.

[58] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, pp. 335–341, 1949.

[59] H. Mozaffari, V. Shejwalkar, and A. Houmansadr, "Every vote counts: Ranking-based training of federated learning to resist poisoning attacks," in *Proc. USENIX SEC 2023*, pp. 17 215–1738.

[60] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proc. ACM CCS 2005*.

[61] F. Neil, *List of Data Breaches and Cyber Attacks in 2023*, 2023, https://www.itgovernance.co.uk/blog/list-of-data-breaches-and-cyber-attacks-in-2023.

[62] 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*.

[63] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," 1999. [Online]. Available: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

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

[65] D. Pasquini, G. Ateniese, and M. Bernaschi, "Interpretable probabilistic password strength meters via deep learning," in *Proc. ESORICS 2020*.

[66] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, "Improving password guessing via representation learning," in *Proc. IEEE S&P 2021*, pp. 265–282.

[67] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Proc. AAAI 2010*, vol. 24, no. 1, pp. 1607–1612.

[68] M. L. Puterman, "Markov decision processes," ser. Handbooks in Operations Research and Management Science, 1990, vol. 2, pp. 331–434.

[69] S. Sahin and F. Li, "Don't forget the stuffing! revisiting the security impact of typo-tolerant password authentication," in *Proc. ACM CCS 2021*, pp. 252–270.

[70] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, pp. 379–423, 1948.

[71] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," in *Proc. ICML 2017*, pp. 1–17.

[72] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[73] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein auto-encoders," in *Proc. ICLR 2018*, pp. 1–16.

[74] B. Ur, F. Alfieri, M. Aung, L. Bauer, N. Christin, J. Colnago, L. F. Cranor, H. Dixon, P. E. Naeini, H. Habib, N. Johnson, and W. Melicher, "Design and evaluation of a data-driven password meter," in *Proc. CHI 2017*. ACM, pp. 3775–3786.

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

[76] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, "Measuring real-world accuracies and biases in modeling password guessability," in *Proc. USENIX SEC 2015*, pp. 463–481.

[77] H. Valizadegan, R. Jin, R. Zhang, and J. Mao, "Learning to rank by optimizing ndcg measure," in *Proc. NeurIPS 2009*, pp. 18 835–1891.

[78] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NeurIPS 2017*, pp. 5998–6008.

[80] R. Veras, C. Collins, and J. Thorpe, "On semantic patterns of passwords and their security impact." in *Proc. NDSS 2014*.

[81] D. Wang, X. Shan, Q. Dong, Y. Shen, and C. Jia, "No single silver bullet: Measuring the accuracy of password strength meters," in *Proc. USENIX SEC 2023*, pp. 947–964.

[82] D. Wang and P. Wang, "The emperor's new password creation policies," in *Proc. ESORICS 2015*, pp. 456–477.

[83] 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.

[84] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM CCS 2016*, pp. 1242–1254.

[85] D. Wang, Y. Zou, Y. Xiao, S. Ma, and X. Chen, "Pass2edit: A multi-step generative model for guessing edited passwords," in *Proc. USENIX SEC 2023*, pp. 983–1000.

[86] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. USENIX SEC 2023*, pp. 965–982.

[87] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. IEEE S&P 2009*, pp. 391–405.

[88] D. Wheeler, "zxcvbn: Low-budget password strength estimation," in *Proc. USENIX SEC 2016*, pp. 157–173.

[89] C. Wissler, "The spearman correlation formula," *Science*, vol. 22, no. 558, pp. 309–311, 1905.

[90] Q. Xie, Z. Dai, E. Hovy, T. Luong, and Q. Le, "Unsupervised data augmentation for consistency training," in *Proc. NeurIPS 2020*.

[91] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han, "Chunk-level password guessing: Towards modeling refined password composition representations," in *Proc. ACM CCS 2021*, pp. 5–20.

[92] M. Xu, J. Yu, X. Zhang, C. Wang, S. Zhang, H. Wu, and W. Han, "Improving real-world password guessing attacks via bi-directional transformers," in *Proc. USENIX SEC 2023*, pp. 1001–1018.

[93] F. Yu and M. V. Martin, "Gnpassgan: Improved generative adversarial networks for trawling offline password guessing," in *Proc. EuroS&PW*, pp. 10–18.

[94] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proc. AAAI 2017*.

[95] V. Zimmermann, "From the quest to replace passwords towards supporting secure and usable password creation," Ph.D. dissertation, Technical University of Darmstadt, Germany, 2021.

Appendix A. Detailed proofs

Here we provide the password characters notation used in this paper in Appendix A.2, detailed proofs for Equation 7 in Appendix A.2 as well as Equation 8 in Appendix A.3, discussion of alternative solutions in Appendix A.4, some supplementary experimental results in Appendix A.5, and further analysis about the crack password in Appendix A.6.

A.1. Notations

Table 9 summarizes the various categories of printable characters and the begin and end symbols used in this paper, which are ASCII 32-126. This is also the standard for token data pre-processing and model training in this work.

TABLE 9: Typical categories of characters in this paper.

Character type	Symbols	Count
Lower letters	a b c d e f g h i j k l m n o p q r s t u w v x y z	26
Upper letters	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	26
Digits	0 1 2 3 4 5 6 7 8 9	10
Special symbol	Space ! " # $ % & ' ( ) * + , – . / : ; 〈 = 〉 ? @ [\ ] ^ _ } { | } ~	33
Begin & End	B_s E_s	2
Total types	ASCII 32-126 and B_s E_s	97

A.2. Proof of the MC rollout converge

As the number of visits to each reward tends to infinity, each estimated reward by Monte Carlo rollout converges. This is easily understood in the context of first-visit MC: each reward is an independently and identically distributed estimate with the finite variance of $\mathcal{Q}_{\theta,\varphi}(\mathrm{PW}_{1:t-1}, \mathcal{E})$, and by the law of large numbers, the sequence of these estimates' means converges to the expected value. Each mean itself is an unbiased estimate, and the standard error of its error decreases as $1/\sqrt{N}$, where N is the number of rewards. Assuming we conduct N trials and obtain N trajectories, denoted as $\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(N)}$, with corresponding total rewards of $\mathcal{Q}_{\theta,\varphi}(\tau^{(1)}), \mathcal{Q}_{\theta,\varphi}(\tau^{(2)}), \ldots, \mathcal{Q}_{\theta,\varphi}(\tau^{(N)})$, the total reward can be approximated as follows:

$$\lim_{N\to\infty} \mathbb{P}\left(\left|\frac{1}{N}\sum_{n=1}^{N} \mathcal{Q}_{\theta,\varphi}(\tau^{(n)}) - \mathcal{Q}_{\theta,\varphi}(\mathrm{PW}_{1:t-1}, \mathcal{E})\right| > \varepsilon\right) = 0. \tag{15}$$

A.3. Proof of the policy gradient

We now prove our proposed policy gradient update algorithm. The goal of reinforcement learning [72] is to learn a policy $\pi_\theta(a_t|s_t)$ that maximizes the expected reward $\mathfrak{L}(\theta) = \mathbb{E}_{x_{1:T}\sim P_\theta(x_{1:T})}\left[\mathcal{T}\left(x_{1:T}\right)\right]$, as mentioned at Section 1.1. One direct method is to search the policy space directly to find the optimal policy, known as policy search. Policy search [45], [67] is essentially an optimization problem and can be categorized into gradient-based optimization and non-gradient-based optimization. Policy search may not require

value functions, but it can directly optimize the policy. The agent needs to explore parameterized policies that can handle continuous states and directly learn policies.

$$\begin{aligned}\frac{\partial \mathfrak{L}(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta}\int P_\theta(\tau)\mathcal{T}(\tau)\mathrm{d}\tau \\ &= \int \left(\frac{\partial}{\partial \theta}P_\theta(\tau)\right)\mathcal{T}(\tau)\mathrm{d}\tau \\ &= \int P_\theta(\tau)\left(\frac{1}{P_\theta(\tau)}\frac{\partial}{\partial \theta}P_\theta(\tau)\right)\mathcal{T}(\tau)\mathrm{d}\tau \\ &= \int P_\theta(\tau)\left(\frac{\partial}{\partial \theta}\log P_\theta(\tau)\right)\mathcal{T}(\tau)\mathrm{d}\tau \\ &= \mathbb{E}_{\tau\sim P_\theta(\tau)}\left[\frac{\partial}{\partial \theta}\log P_\theta(\tau)\mathcal{T}(\tau)\right],\end{aligned} \tag{16}$$

where $\frac{\partial}{\partial \theta}\log P_\theta(\tau)$ represents the partial derivative of the function $\log P_\theta(\tau)$ with respect to θ. From Equation 16, it can be seen that the direction in which parameter θ is optimized is such that it increases the probability $P_\theta(\tau)$ of trajectories τ with higher total rewards $\mathcal{T}(\tau)$. Meanwhile, $\frac{\partial}{\partial \theta}\log P_\theta(\tau)$ could be partitioned into:

$$\begin{aligned}\frac{\partial}{\partial \theta}\log P_\theta(\tau) &= \frac{\partial}{\partial \theta}\log\left(P_\theta(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t \mid s_t)\,P_\theta(s_{t+1}\mid s_t,a_t)\right) \\ &= \frac{\partial}{\partial \theta}\left(\log P_\theta(s_0)+\sum_{t=0}^{T-1}\log \pi_\theta(a_t\mid s_t)+\sum_{t=0}^{T-1}\log P_\theta(s_{t+1}\mid s_t,a_t)\right) \\ &= \sum_{t=0}^{T-1}\frac{\partial}{\partial \theta}\log \pi_\theta(a_t\mid s_t).\end{aligned} \tag{17}$$

The derivative *w.r.t.* θ of $P_\theta(s_0)$ and $P_\theta(s_{t+1}\mid s_t,a_t)$ are equal to zero. Thus, it becomes evident that the gradient $\frac{\partial}{\partial \theta}\log P_\theta(\tau)$ is solely dependent on the policy function, and not on the state transition probabilities. As a result, the policy gradient, denoted as $\frac{\partial \mathfrak{L}(\theta)}{\partial \theta}$, can effectively be expressed using the following mathematical formulation:

$$\begin{aligned}\frac{\partial \mathfrak{L}(\theta)}{\partial \theta} &= \mathbb{E}_{\tau\sim P_\theta(\tau)}\left[\left(\sum_{t=0}^{T-1}\frac{\partial}{\partial \theta}\log \pi_\theta(a_t\mid s_t)\right)\mathcal{T}(\tau)\right] \\ &= \mathop{\mathbb{E}}_{\mathrm{PW}_{1:T}\sim \mathcal{G}_\theta}\left[\sum_{t=1}^{T}\sum_{a_t\in\mathbb{A}}\nabla_\theta \log \pi_\theta(a_t|\mathrm{PW}_{1:t-1})\delta_t\right].\end{aligned} \tag{18}$$

A.4. Discussion of alternative solutions

Statistical approaches. Using PCFG [87], the complex structures in training datasets lead to low generation probabilities and an overestimation of security strength. PCFG [87] lacks an effective method to handle unseen basic structures, further complicating accurate generation as a larger guess number. The effectiveness of a Markov model in password cracking peaks at a fourth-order model, as higher orders lead to overfitting due to data sparseness and the dependency on the frequency of *n-gram* strings in the training dataset. To mitigate this issue, Ma *et al.* [60] recommend Laplace smoothing which adds a small constant to character frequencies post-training, yet this method mainly lessens rather than resolves the sparseness problem and overlooks crucial sub-string features. TarGuess-I [84] and TarMarkov [82] are variants of previous models.

Machine Learning approaches. RFGuess [86] constructs a sequence model using n-grams as classification features to predict the next tokens, while FLA [56] utilizes RNNs for feature extraction using n-grams to predict the next ones. CPG/DPG [66] constructs password guessing models. GANs and VAEs primarily focus on data generation and reconstruction, and do not directly involve the complexities of environmental interaction or objective optimization. GANs and VAEs need to fully map data into the latent space, but they struggle to learn the sequential relationships between passwords. PassBERT [26] employs BERT as the base model for password guessing but faces shortcomings such as slow inference speed and its inadequacy for generation tasks. Conversely, CWAE [66] is limited to conditional password guessing as it optimizes the entire password as a whole, which complicates sequence generation and leverages additional advantages, like PII or partial password.

LLM-based approaches. We have observed that large language models (LLMs) excel in natural language generation tasks. However, our experiments suggest that this may not apply to password guessing. As discussed in Section 2, passwords differ significantly from natural language; they are typically short and have a vocab of only 96 characters, compared to the 100K-prompt range found in natural language. Furthermore, while the goal of LLMs is to align with human preferences (RLHF) [21], the objective of password guessing aligns with learning to rank, providing crucial insight for our research. Even RLHF, as employed by LLM, relies on RL techniques as ours (such as PPO [20], [30], [43], *etc.*) to boost performance. However, according to Occam's principle, we prioritize simplicity. Thus, we have opted for a solution with fewer parameters. Our main task is to solve a learning to rank *alignment* problem. Notably, a significant challenge with LLMs lies in their slow inference speed and difficulty in generating a large number of password guesses.

A.5. Supplementary experimental results

We considered 14 more guess experiments of cross-site password guessing scenarios. A **bold** value in each row means that it is the highest one in Table 17. More results focusing on trawling password guessing can be found in Table 16, along with supplementary experiments on conditional password guessing in Table 7.

Comparison with other RL methods. We also noticed that some precious works have also used RL techniques to trawling guessing, such as [18]. Our approach differs from other RL methods as we address more of learning-to-rank alignment problem rather than directly optimizing for password guessing. Password guessing requires ranking the

TABLE 10: Comparison of conditional password guessing models with the other SOTA models: PassBERT [92] and CWAE [66].†

Experimental setup		RankGuess-Mask	Pass-BERT [92]	CWAE [66]
Guessing scenario	Guess number			
CSDN → LinkedIn	10	28.12%	25.80%	29.13%
	10^4	59.80%	56.13%	61.19%
	10^7	75.86%	71.88%	85.19%
CSDN → 000Webhost	10	18.59%	16.82%	15.44%
	10^4	51.79%	49.86%	44.56%
	10^7	54.01%	53.61%	50.58%
Wishbone → Dodonew	10	24.14%	23.44%	19.19%
	10^4	56.00%	55.69%	44.34%
	10^7	67.65%	64.78%	57.58%
Wishbone → Taobao	10	30.17%	25.11%	27.70%
	10^4	60.29%	56.42%	52.14%
	10^7	73.77%	67.63%	64.37%
Rockyou → LinkedIn	10	23.12%	13.38%	16.06%
	10^4	61.77%	58.16%	44.01%
	10^7	78.52%	62.35%	72.69%
Rockyou → Clixsense	10	26.91%	25.12%	24.84%
	10^4	68.01%	58.69%	55.87%
	10^7	80.13%	66.99%	70.19%

† A value with dark gray (resp. light gray) represents the highest one (resp. 2nd one).

guess candidates, which has a clear preference, as described in Section 1. Previous methods (e.g., [18]) did not address the core challenge of password guessing (i.e., learning-to-rank) and merely solved learning-to-generate guessing tasks. Here, we provide a comparison of them in Table 12.

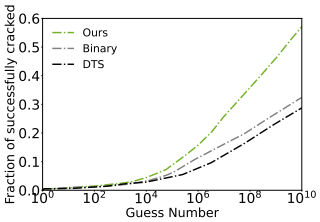


Figure 12: 1W Twitter →CSDN

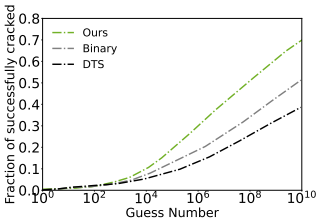


Figure 13: 1M Twitter →Taobao

Figures 12 and 13 show that learning-to-rank significantly enhances password guessing, offering substantial gains over the traditional binary (true/false for classifying password generated by generator) approach. Password guessing requires ranking the guess candidates, which has a clear preference. There are mainly three differences with those adversarial password generation models: (1) The Q-network described in [18] is used solely for Dynamic Temperature Sampling (DTS); (2) The other adversarial methods (e.g., PassGAN [36], GENPass [52]) fail to address learning-to-rank and merely focus on learning-to-generate; (3) In contrast, RankGuess builds a preference model to align with the preferences of guessing. RankGuess transforms the reward from a single feedback signal into preference feedback (e.g., $f(q, pw_1) \geq f(q, pw_2) \geq \cdots \geq f(q, pw_n)$), where q is the additional information. Furthermore, their method is somewhat limited and cannot be applied to a broader range of guessing scenarios in-the-wild.

TABLE 11: The guessing performance of our RankGuess model is compared using Cosine, Manhattan, and Euclidean metrics.

Scenarios	Cosine	Manhattan	Euclidean
0.01M Twitter → CSDN	57.92%	49.58%	29.73%
50%PII-ClixSense→50%PII-12306	99.29%	89.01%	81.45%
Rockyou(Con.) → LinkedIn(Con.)	78.52%	64.95%	52.23%

Comparison with GAN-based methods. As shown in Figures 5 and 6, GAN-based models perform less effectively than alternative existing models like PCFG [87], RFGuess [86], and FLA [56]. As Pasquini et al. [66] pointed out, *PassGAN requires up to ten times more guesses to reach the same number of matched passwords as its probabilistic and non-full probabilistic competitors*. In reality, non-full probabilistic models (e.g., PassGAN [36], GNPassGAN [93], and WGAN [66]) typically generate password samples by sampling from random noise, without producing an explicit probability value for guess candidates. This process is filled with randomness and uncertainty, making it unable to prioritize passwords that $\mathcal{A}$ considers to have a higher likelihood, leading to a large number of duplicates [36], [66], [93]. Here we present comparison results under the same experimental setups: we compare with the previous GAN-based model *using a large number of guesses*. We trained on 50% Rockyou and tested on Twitter in Table 12.

TABLE 12: The performance of our RankGuess in comparison with GAN-based models (i.e., PassGAN [36] and WGAN [66]) training on 50% Rockyou to crack the Twitter at $\geq 10^8$ guesses).

Guess number	PassGAN [36]	WGAN [66]	RankGuess
$1 \cdot 10^8$	6.31%	8.67%	**52.12%**
$1 \cdot 10^9$	17.62%	22.53%	**60.51%**
$1 \cdot 10^{10}$	27.97%	35.49%	**70.67%**
$5 \cdot 10^{10}$	35.01%	46.37%	**73.34%**

Why choose cosine similarity? For Equation 5 using cosine based similarity, when data differences are concentrated in a few dimensions, the *L1 norm* (Manhattan) can provide better performance. *L2 norm* (Euclidean) might require normalization when the scales vary. The cosine-based metric performs the best, because (1) passwords may have different lengths but can still be quite similar; and (2) cosine-based metric focuses on both the angle between two vectors and their lengths, making it effective for handling sparse password vectors [64], [85]. We have verified this with experiments in Table 11. The performance of the cosine-based metric surpasses that of the Manhattan and Euclidean methods, with improvements ranging from 8.34% $\sim$ 28.19%.

RankGuess-PSM for administrators. We have set up a toy usage example for website administrators or public safety maintainers, who can deploy our model in the real-world website as shown below to achieve the desired effect for users to prevent easy-to-guess passwords in Figure 15.

The model is saved in binary format, compressed to about 1.34 MB, and can be deployed offline via methods such as web frontends and browser plugins. Additionally,

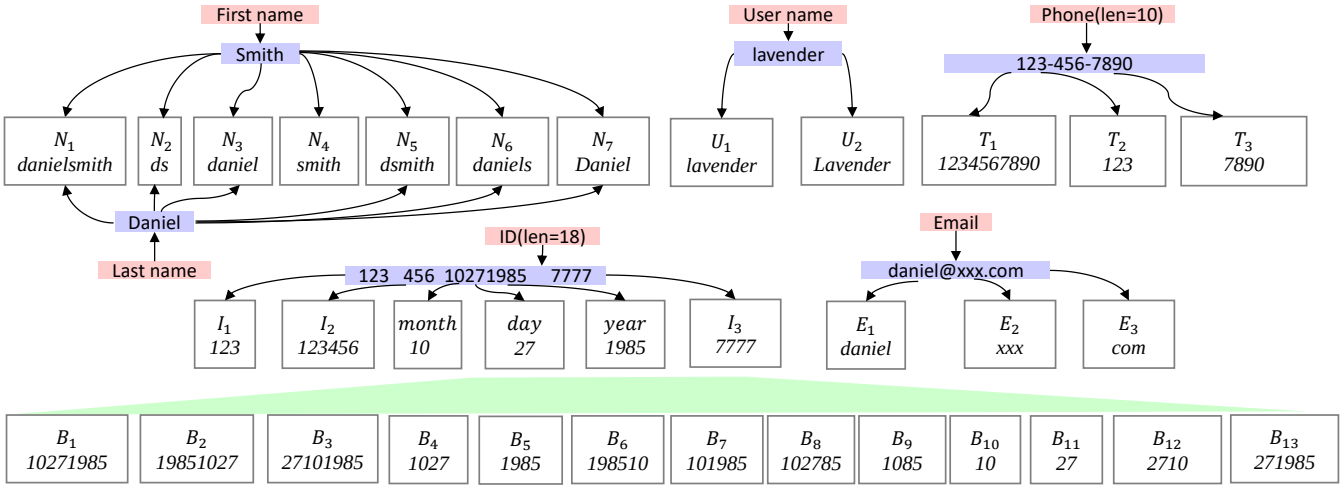


Figure 14: An example of a process where a user's password is matched with their PII used by themselves is detailed. We have summarized the specific rules in Table 2. For instance, consider the targeted password `daniel1027`. It can be represented in different ways: $\{\mathbf{N}_3\,\mathbf{B}_4\}$, $\{\mathbf{N}_3\,1,0,2,7\}$, $\{\mathbf{N}_3\,\mathbf{B}_{10},2,7\}$,$\{\mathbf{N}_3\,1,0,\mathbf{B}_{11}\}$,$\{\mathbf{N}_3\,\mathbf{B}_{10}\,\mathbf{B}_{11}\}$,$\{\mathbf{E}_3\,\mathbf{B}_4\}\ldots$, and $\{\mathrm{d},\mathrm{a},\mathrm{n},\mathrm{i},\mathrm{e},\mathrm{l},1,0,2,7\}$, a total of 17 representations.

a pre-generated dictionary for MC simulations, containing 100,000 passwords and their probabilities (approximately 800KB), is required. The total storage needed for the password strength meter application is roughly 3MB. Note that this usage method is not ***set in stone***. It can be customized to meet more advanced requirements for the website. The design of levels could be refined for finer decision-making and more detailed feedback [81]. For example, an additional feature could be added to provide suggestions for strengthening passwords by identifying and alerting users about insecure elements. An extra module could also provide PII checking to ensure passwords comply with specific policies. The RANKGUESS MODULE in Figure 15 should be replaced with RANKGUESS-PII to tackle the PII-taggs.

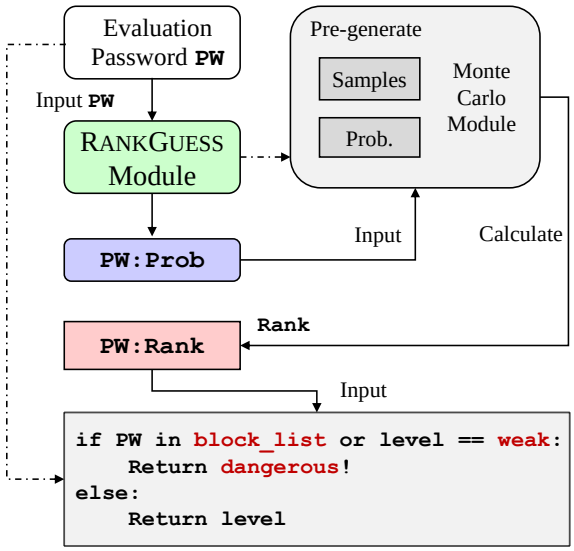


Figure 15: Example of RANKGUESS-PSM for strength evaluation.

Why remove passwords with len>30? As with the major studies [80], [83], [86], we remove password strings with len>30, because after manually scrutinizing the original datasets, we find that these overly long strings do not seem to be generated by users, but more likely by password managers or simply junk information. Generally, such unusually long passwords are often beyond the scope of the adversary who care about cracking efficiency [12]. Besides, the fraction of such overly long passwords is negligible (i.e., $\ll 1\%$ for all 12 datasets as shown in Table 3), which will not affect the experimental results and our conclusions.

Against randomly generated passwords. In this part, we evaluate and analyze the strength of random passwords generated by managers. Specifically, we use the JavaScript-based random password generators from popular password managers including, including 1Password [57], LastPass [44], and Bitwarden [11]. We generate 1 million (M) random passwords from each manager using the 3class8 criteria (i.e., length 8 with three character classes) *for a fair comparison*. We use the RANKGUESS and MC method [24] to measure the strength of these random passwords.

TABLE 13: Guessing performance of our RANKGUESS training using Rockyou train data in comparison with random passwords.

Guess number	1Password [57]	LastPass [44]	Bitwarden [11]
$1\cdot10^{8}$	0.00%	0.00%	0.00%
$1\cdot10^{10}$	0.00%	0.00%	0.00%
$1\cdot10^{12}$	0.00%	0.00%	0.01%
$1\cdot10^{14}$	0.26%	0.15%	0.36%
$1\cdot10^{16}$	6.83%	6.74%	9.89%
$1\cdot10^{18}$	59.18%	66.23%	69.39%
$1\cdot10^{20}$	98.69%	100.00%	100.00%

In Table 13, at the early password guessing stage ($10^{8}\sim10^{12}$ guesses), all password managers show 0.00% guess success, indicating strong resistance against initial brute-force attempts. At mid-range guessing stage

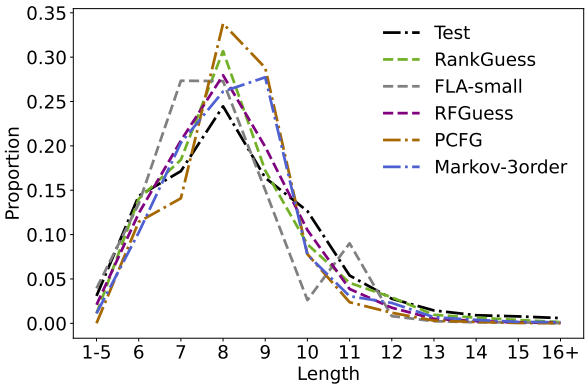


Figure 16: The distributions of password lengths cracked by each model. Our RANKGUESS best approximates the length distributions of the test set among the five password guessing models.

(10^{12}∼10^{18} guesses), Bitwarden [11] shows slightly higher guess success rates compared to 1Password [57] and LastPass [44], suggesting it might be slightly more vulnerable at this stage. At large guess number (>10^{18} guesses), all password managers become significantly more vulnerable. It can be found that Bitwarden [11] generates the weakest passwords, followed by LastPass [44]. The strongest passwords are generated by 1Password [57]. This conclusion is consistent with the findings in [62], *where we assume the adversary $\mathcal{A}$ does not have any knowledge about PII or password policy 3class8, which differs from [62].*

Detailed information about our semantic dictionaries. To make our work as reproducible as possible and to facilitate the community, we now detail how to construct our semantic-based dictionaries in Figure 10. The first dictionary "English word" from the Oxford English Dictionary (OED) via https://www.oed.com/ includes 1M common lowercase English words. The second dictionary "English name" is a combination of the "firstname" and "lastname" featuring 1M commonly used English names from the American National Corpus (ANC) via the link https://anc.org/. The third dictionary pinyin words are from the Chinese Hanyu Pinyin Database https://www.cjk.org/data/chinese/nlp/chinese-hanyu-pinyin-database/. The last complex keyboard pattern is created by systematically identifying patterns in keyboard sequences that are more than three characters and extend in various directions: *horizontal, vertical, diagonal, and their reverse*.

A.6. Further analysis

Length distribution analysis. Here we use the passwords cracked by each of these five models to explore their differences in cracking passwords of varied lengths. Figure 16 shows that our model primarily focuses on passwords with lengths of 6 and 8. Initially, it is evident that RANKGUESS achieves a higher success rate for passwords of length 6 compared to other models (i.e., FLA-small [56], RFGuess [86], PCFG [87], and Markov-3order [60]), exceeding them by 0.56%. Specifically, FLA-small [56], achieves a rate of 13.47%, RFGuess [86] 12.39%, PCFG [87] 11.43%, and Markov-3order [60] 10.23%. Furthermore, for passwords exceeding 10 characters, RANKGUESS successfully cracks (%) $8.91 + 4.56 + 2.93 + 0.96 + 0.64 + 0.38 + 0.14$ (18.52%) of passwords. In contrast, FLA-small [56] cracks 13.94%, determined by $2.62 + 9.02 + 0.81 + 0.24 + 0.12 + 0.11 + 0.02$ (%), while RFGuess [86] cracks 17.08%, calculated as $10.51 + 3.87 + 1.76 + 0.51 + 0.26 + 0.10 + 0.07$ (%). Our RANKGUESS demonstrates a notable advantage in cracking longer passwords, offering a 1.44% improvement over other models. In the case of passwords of length 6, the cracking rate of RANKGUESS is higher than that of other models, making it easier to discover weak passwords. For passwords longer than 10 characters, RANKGUESS exhibits a higher proportion, making it easier to crack long passwords. As password length increases, other models show a significant decrease in cracking performance for these passwords, while RANKGUESS continues to exhibit similarities to the length distribution of the test set (such as at password lengths 11 and 16), indicating that RANKGUESS not only performs exceptionally well on shorter passwords but also remains competitive on longer passwords. Moreover, we use the total variation distance (TVD) in Figure 16 to measure of the difference between test distribution $P(x)$ and crack model distribution $Q(x)$, which can be defined as:

$$\mathrm{TVD}(P, Q) = \frac{1}{2} \sum_{x \in X} |P(x) - Q(x)| \tag{19}$$

The TVD between the length distributions of the test set and RANKGUESS is 0.0848, which is less than between the test set and RFGuess [86] at 0.1243, and the test set and FLA-small [56] at 0.2051, the test set and PCFG [87] at 0.2772, and the test set and Markov-3order [60] at 0.2118. The cracked password length distribution between RANKGUESS and the test set is the remarkably closest, meaning that the passwords generated by our RANKGUESS *significantly* approximate the test set.

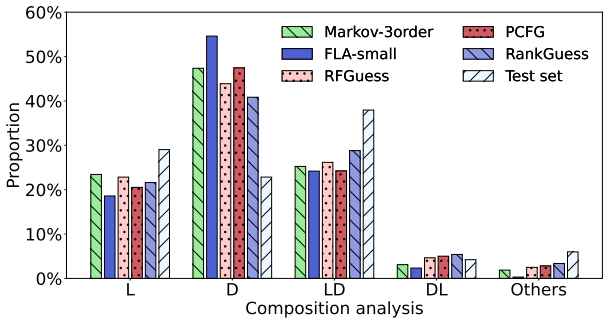


Figure 17: The composition distributions of cracked passwords.

Password composition analysis. Previous research [83] demonstrated that the three most common structural patterns overwhelmingly represent a significant portion of user passwords. For instance, in Chinese datasets, the top three structures: D, LD, and DL comprise, on average, 81.90% of passwords, while in English datasets, the patterns L, LD, D

TABLE 14: Ten examples of password cracked by each model: RANKGUESS, FLA-small [56], PCFG [54], Markov-3order/4-order [60] and RFGuess [86]. We summarize the number of guesses each model needs to crack the password. RANKGUESS has the smallest number of guesses for these passwords among all the models. The smallest number is highlighted in **bold**, indicating the best performance. An ∞ value in each row means that the PCFG [87] cannot learn the templates and filled content from these training sets, resulting in an inability to guess the corresponding passwords. This is due to an intrinsic flaw in the nature of the PCFG [87].

Index	Password	RANKGUESS	FLA-small [56]	PCFG [87]	Markov-3order [82]	Markov-4order [82]	RFGuess [86]
1	@@969810	$\mathbf{1.93 \times 10^9}$	5.87×10^{11}	∞	1.21×10^{13}	5.27×10^{15}	4.94×10^{10}
2	apple@416	$\mathbf{1.08 \times 10^{10}}$	2.03×10^{11}	∞	1.37×10^{13}	6.44×10^{12}	4.51×10^{11}
3	GS880511	$\mathbf{2.10 \times 10^9}$	5.92×10^{10}	8.29×10^{13}	3.25×10^{12}	3.59×10^{12}	1.55×10^{11}
4	6064915+	$\mathbf{1.02 \times 10^9}$	8.16×10^9	2.36×10^{14}	1.20×10^{13}	6.65×10^{14}	4.23×10^{13}
5	administrator	5.35×10^{11}	5.23×10^9	5.82×10^{12}	1.37×10^{11}	$\mathbf{6.81 \times 10^5}$	1.13×10^8
6	13561059393	4.01×10^{10}	9.96×10^8	∞	2.13×10^{11}	1.84×10^{10}	$\mathbf{7.03 \times 10^8}$
7	fenghuanwuqi	$\mathbf{4.38 \times 10^{10}}$	5.82×10^{10}	∞	6.53×10^{11}	1.29×10^{12}	3.89×10^{11}
8	kingsakura	$\mathbf{7.93 \times 10^9}$	9.32×10^{10}	8.30×10^{16}	1.67×10^{10}	4.97×10^{11}	4.01×10^{10}
9	liqingming100	$\mathbf{1.08 \times 10^{10}}$	6.19×10^{10}	∞	5.45×10^9	2.01×10^9	1.25×10^{11}
10	127789777	$\mathbf{1.17 \times 10^8}$	7.02×10^{12}	4.39×10^{14}	2.31×10^{12}	2.73×10^{13}	1.37×10^{11}

make up 81.26%. Based on these findings, we categorize the passwords that were successfully cracked into five distinct groups: L (lower-case sequences), D (digit sequences), LD, DL, and others. Figure 17 shows that different guessing models are good at cracking passwords with different structures. From Figure 17, we can see that FLA-small [56] and Markov-3order [60], both n-gram-based sequence models, are particularly adept at cracking passwords composed entirely of numbers, with the D structure being 54.61% and 47.36%, respectively. In contrast, RANKGUESS has a lower capability in this structure, at only 41.83%. In terms of pure letters, Markov-3order [60] and RFGuess [86] are stronger, at 22.43% and 22.82%, respectively, with RANKGUESS being weaker by 1.22% than RFGuess [86]. However, in the LD structure, RANKGUESS has a cracking percentage of 28.80%, which is 2.64% higher than RFGuess [86] at 26.16% and Markov-3order [60] at 25.24%. In the DL structure, RANKGUESS cracking percentage is 5.40%, which is 0.39% higher than PCFG [87]. Additionally, in other structures, RANKGUESS exceeds PCFG [87] by 0.56%.

Examples of password cracked by each model. We provide ten representative examples of passwords cracked independently by each model in Table 15. FLA-small [56] excels in deciphering passwords comprised entirely of numerical sequences, such as `1512007446`, `1505452412`, and `1366303655`, indicating its proficiency in handling numeric passwords. Conversely, RFGuess [86] demonstrates its strength by cracking passwords that incorporate common phrases, cultural references, or popular elements, including `Forever?`, `administrator`, `Beijing2008`, and `infosec`. Our RANKGUESS model shows a clear preference for complex structures; for instance, it successfully cracks passwords like `139389.x`, which combines numbers with a period. It also adeptly handles passwords with higher complexity and cultural significance, such as `zxh_0606` (a mix of letters and numbers with an underscore), `890522Tc` (potentially a date combined with letters), and `Wqh@1227` (a blend of letters, numbers, and special characters), which may require sophisticated attempts.

TABLE 15: Ten examples of password cracked independently by each model: RANKGUESS, FLA-small [56] and RFGuess [86].

Index	RANKGUESS	FLA-small [56]	RFGuess [86]
1	139389.x	1512007446	Forever?
2	!(*^$@^(){}<	wodecsd	administrator
3	zxh_0606	jia123000	Beijing2008
4	890522Tc	1505452412	12345654321
5	Wqh@1227	1366303655	power870
6	xiaoqiufeng520	102932.	hellodearbook
7	jtr520xy	1381082706	pass@word
8	resender19821111	jngt200	infosec
9	w1988423*	0.03214	daniel1988
10	zbzb@880923	1314?131?	piggy110

Guess number of password cracked by each model. Each model exhibits significant differences in the number of guesses required for passwords of varying structures and complexities. For instance, the RANKGUESS, RFGuess [86] and FLA-small [56] models generally need fewer guesses in most cases, while the PCFG [87] shows an infinite number of guesses for several samples, indicating its ineffectiveness for these types of passwords. The `administrator` is very inefficiently guessed by the Markov-4order [60] ($6.81*10^5$ guesses), likely because it is a common dictionary word. Conversely, the password `kingsakura` requires a high number of guesses ($8.30 * 10^{16}$) in the PCFG [87], demonstrating the model's limited capability in handling this compound vocabulary. In all, our RANKGUESS requires fewer guessing attempts to crack these passwords.

TABLE 16: Comparison of the cracking rate of five different trawling password guessing methods*

Experimental setup		RankGuess	PCFG [87]	Markov-3order [82]	Markov-4order [82]	RFGuess [86]	FLA-small [56]
Attacking scenario	Guess Number						
#1:1M CSDN→CSDN_rest	10^3	11.31%	**15.82%**	10.05%	13.3%	14.83%	6.98%
	10^6	27.21%	32.03%	25.15%	28.74%	**30.42%**	14.98%
	10^9	**58.72%**	42.51%	55.5%	54.58%	50.02%	46.94%
	10^{12}	**82.95%**	46.02%	80.12%	75.07%	69.95%	77.37%
	10^{15}	**92.11%**	47.01%	89.29%	85.95%	87.04%	90.36%
	10^{18}	**98.24%**	47.01%	96.17%	94.41%	95.41%	96.94%
#2:1M Twitter→Twitter_rest	10^3	4.54%	4.62%	1.63%	3.32%	**5.07%**	4.84%
	10^6	30.79%	28.51%	18.64%	25.52%	**31.33%**	29.02%
	10^9	**61.33%**	43.37%	53.07%	51.01%	58.81%	55.75%
	10^{12}	**84.76%**	48.82%	77.34%	72.52%	82.46%	79.41%
	10^{15}	**94.65%**	50.31%	88.77%	86.24%	92.71%	91.06%
	10^{18}	**97.51%**	51.26%	93.85%	93.31%	96.75%	95.37%
#3:1M LinkedIn→Taobao	10^0	1.13%	1.51%	1.51%	1.51%	**1.51%**	0.52%
	10^2	2.21%	**3.28%**	1.62%	2.49%	3.05%	2.26%
	10^4	4.79%	**5.32%**	3.32%	4.41%	5.06%	4.98%
	10^6	**11.49%**	8.83%	7.32%	8.35%	10.46%	9.25%
	10^8	**27.03%**	13.52%	23.45%	23.33%	25.11%	24.13%
	10^{10}	**46.63%**	17.93%	41.95%	38.44%	43.24%	43.41%
#3:1M Taobao→000Webhost	10^0	0.84%	**1.94%**	0.15%	1.42%	1.37%	0.15%
	10^2	2.74%	**4.75%**	2.79%	4.01%	4.75%	2.48%
	10^4	7.33%	**9.29%**	5.54%	6.75%	8.60%	6.78%
	10^6	16.73%	**18.95%**	14.35%	16.31%	18.21%	16.04%
	10^8	**37.27%**	30.09%	35.21%	37.06%	37.17%	33.89%
	10^{10}	**65.82%**	38.16%	64.40%	60.87%	62.93%	60.71%
#5:1M CSDN→Taobao	10^0	**1.28%**	0.45%	0.45%	0.45%	0.67%	1.05%
	10^2	3.36%	2.76%	2.98%	3.62%	**3.74%**	2.92%
	10^4	**8.62%**	7.56%	5.75%	7.18%	7.63%	4.16%
	10^6	**17.12%**	15.23%	14.47%	16.06%	15.46%	14.63%
	10^8	**37.25%**	22.75%	33.32%	32.95%	32.95%	33.01%
	10^{10}	**64.11%**	29.82%	59.5%	53.84%	57.69%	59.65%
#6:0.01M Twitter→Rockyou	10^0	0.61%	**1.08%**	0.12%	0.66%	0.33%	0.24%
	10^2	2.74%	**2.77%**	1.14%	1.93%	2.17%	1.75%
	10^4	**8.93%**	8.27%	3.86%	6.64%	5.19%	7.60%
	10^6	**26.26%**	13.28%	16.18%	14.06%	20.64%	22.03%
	10^8	**47.21%**	14.18%	30.06%	24.75%	42.26%	40.63%
	10^{10}	**70.69%**	14.73%	46.24%	40.81%	63.93%	57.72%
#7:0.1M Twitter→Rockyou	10^0	0.48%	**0.96%**	0.18%	0.96%	0.48%	0.48%
	10^2	3.36%	**3.45%**	1.37%	1.97%	3.06%	2.94%
	10^4	12.46%	**12.82%**	4.62%	10.35%	12.16%	11.98%
	10^6	**35.13%**	26.57%	22.95%	25.72%	31.33%	29.41%
	10^8	**57.99%**	33.25%	46.58%	41.15%	53.46%	54.25%
	10^{10}	**76.74%**	34.69%	64.13%	56.89%	71.79%	72.27%
#8:1M Twitter→Rockyou	10^0	0.20%	**0.88%**	0.32%	0.74%	0.44%	0.32%
	10^2	**3.58%**	3.44%	1.47%	1.95%	3.44%	3.51%
	10^4	16.46%	**17.61%**	4.58%	11.36%	17.48%	16.39%
	10^6	**39.86%**	39.46%	23.64%	32.87%	38.85%	37.22%
	10^8	**64.90%**	48.74%	52.14%	54.31%	60.42%	58.85%
	10^{10}	**83.55%**	53.07%	72.48%	69.23%	76.83%	78.32%

* A **bold** value in each row means that it is the highest one among all six methods. RankGuess performs better when using larger guess numbers.

TABLE 17: Comparison of the cracking rate of five different targeted password guessing methods*

Experimental setup		RankGuess-PII	RFGuess-PII [86]	TarGuess-I [84]	TarMarkov-3order [82]	TarMarkov-4order [82]
Attacking scenario	Guess Number					
#1: 1M PII-000Web.→PII-PII-000Web._rest	10	**8.36%**	5.78%	8.12%	1.95%	1.98%
	10^5	13.99%	**18.18%**	14.32%	13.08%	13.53%
	10^8	**23.38%**	23.22%	20.59%	22.16%	22.20%
	10^{14}	**73.06%**	62.52%	36.43%	51.46%	53.32%
#2: 1M PII-CSDN → PII-CSDN_rest	10	14.36%	**24.20%**	14.55%	16.47%	15.55%
	10^5	**34.51%**	33.45%	34.11%	34.08%	34.48%
	10^8	**58.69%**	58.06%	34.08%	47.18%	48.92%
	10^{14}	**96.55%**	91.44%	43.67%	77.29%	79.99%
#3: 1M PII-12306 → PII-12306_rest	10	12.53%	**16.32%**	11.28%	8.53%	8.54%
	10^5	**33.29%**	33.21%	29.88%	31.70%	31.56%
	10^8	57.92%	**59.97%**	34.11%	49.82%	46.07%
	10^{14}	**96.68%**	82.40%	44.86%	88.01%	83.43%
#4: 1M PII-Rootkit → PII-Rootkit_rest	10	5.57%	**7.24%**	6.18%	4.91%	4.92%
	10^5	**33.22%**	23.21%	22.05%	20.21%	20.21%
	10^8	**42.78%**	42.54%	28.66%	29.98%	30.01%
	10^{14}	**89.37%**	82.40%	31.34%	69.89%	70.12%
#5: 1M PII-Clix.→PII-Clix._rest	10	8.36%	7.14%	**9.14%**	8.16%	8.11%
	10^5	**35.16%**	31.79%	27.55%	29.23%	30.25%
	10^8	**60.73%**	57.21%	31.63%	51.32%	52.43%
	10^{14}	**96.36%**	86.16%	56.93%	79.90%	84.21%
#6: PII-Mixed†: 90% → 10%	10	**18.07%**	15.36%	17.13%	16.97%	17.73%
	10^5	**31.91%**	29.50%	26.32%	27.89%	28.66%
	10^8	67.86%	**68.32%**	66.62%	63.69%	61.88%
	10^{14}	**99.28%**	96.36%	93.19%	91.82%	92.96%
#7: PII-Mixed: 70% → 30%	10	**16.55%**	15.36%	14.1%	13.94%	14.72%
	10^5	28.70%	27.60%	**29.31%**	26.63%	27.47%
	10^8	**62.67%**	60.17%	61.43%	57.58%	58.85%
	10^{14}	**93.60%**	91.25%	65.66%	88.23%	89.15%
#8: PII-Mixed: 50% → 50%	10	**16.17%**	15.16%	14.32%	12.86%	13.48%
	10^5	27.41%	26.78%	**28.76%**	26.90%	27.57%
	10^8	**57.16%**	56.44%	58.91%	49.77%	50.87%
	10^{14}	**91.73%**	89.23%	61.27%	85.50%	86.79%
#3A: $\frac{1}{2}$ PII-12306 → PII-CSDN	10	**11.88%**	11.28%	12.66%	10.28 %	10.30%
	10^5	**24.69%**	23.56%	22.13%	20.78%	21.53%
	10^8	**47.19%**	45.48%	46.16%	41.36%	40.19%
	10^{14}	**81.50%**	78.13%	62.64%	75.66%	76.96%
#3B: $\frac{1}{4}$ PII-12306 → PII-CSDN	10	**10.31%**	9.73%	9.87%	8.52%	7.61%
	10^5	**18.68%**	17.35%	18.39%	17.10%	15.39%
	10^8	**41.92%**	38.34%	40.32%	38.61%	36.53%
	10^{14}	**72.36%**	69.63%	58.98%	61.94%	60.73%
#3C: $\frac{1}{8}$ PII-12306 → PII-CSDN	10	**7.12%**	7.46%	7.65%	5.27%	6.71%
	10^5	**14.66%**	14.58%	13.79%	12.13%	12.78%
	10^8	**26.77%**	26.16%	24.11%	23.33%	22.66%
	10^{14}	**58.02%**	56.21%	49.66%	55.44%	51.14%
#3A: $\frac{1}{2}$ PII-ClixSense → PII-000Webhost	10	**9.47%**	9.13%	8.56%	8.18%	7.93%
	10^5	**18.01%**	17.49%	16.60%	16.30%	15.85%
	10^8	**23.89%**	23.12%	24.45%	22.40%	21.75%
	10^{14}	**62.50%**	58.88%	53.56%	55.05%	59.62%
#3B: $\frac{1}{4}$ PII-ClixSense → PII-000Webhost	10	8.51%	7.28%	**8.98%**	6.24%	7.18%
	10^5	**16.07%**	15.65%	15.39%	12.38%	14.90%
	10^8	**23.92%**	22.48%	22.59%	20.70%	23.07%
	10^{14}	**58.36%**	55.67%	52.09%	50.01%	51.48%
#3C: $\frac{1}{8}$ PII-ClixSense → PII-000Webhost	10	**8.48%**	7.39%	8.12%	8.01%	8.31%
	10^5	**14.66%**	13.76%	12.91%	13.20%	14.63%
	10^8	**25.21%**	24.94%	24.16%	21.03%	23.38%
	10^{14}	**57.02%**	55.20%	48.93%	52.23%	52.54%

* We considered 14 guess experiments of cross-site scenarios. A **bold** value in each row means that it is the highest one among all five methods.

† We mix three English datasets (*i.e.*, 000Webbhost, Rootkit, and ClixSense) and three Chinese datasets (*i.e.*, CSDN, 12306, and Dodonew) and use 90%, 70% and 50% of them for training and the rest 10%, 30% and 50% for testing. Further, elucidates the stability in the case of a larger training dataset.

Appendix B.
Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

The paper introduces RANKGUESS, a new password-guessing framework utilizing adversarial ranking and reinforcement learning. By framing password guessing as a Markov Decision Process, the framework models the guessing process as a sequence of decisions, making it applicable to three specific scenarios: password trawling, targeted guessing using PII, and partial password guessing. The authors evaluated RANKGUESS on several large datasets, demonstrating that it outperforms existing state-of-the-art methods.

B.2. Scientific Contributions

- Creating a new tool to enable future password security research.
- Addressing an interesting question: guessing with reinforcement learning, a new approach in password security.
- Providing a successful method to derive adversarial ranking in password security.

B.3. Reasons for Acceptance

1) The paper provides a new solution to a well-known challenge in the password-guessing field. The use of reinforcement learning and adversarial ranking significantly improves password guessing, particularly in trawling and PII-aided scenarios.
2) The RANKGUESS framework can potentially be adapted for other password-guessing contexts and enables broader research applications, offering researchers a new tool to assess and improve password security.
3) The framework outperforms state-of-the-art models on multiple datasets. The analysis demonstrated the viability of applying adversarial ranking and a reinforcement learning framework to this long-standing problem.