How to Attack and Generate Honeywords

Ding Wang, Yunkai Zou, Qiying Dong
Nankai University
{wangding, zouyunkai, dqy}@nankai.edu.cn

Yuanming Song
Peking University
ymsong@pku.edu.cn

Xinyi Huang
Fujian Normal University
xyhuang81@gmail.com

***Abstract*—Honeywords are decoy passwords associated with each user account to timely detect password leakage. The key issue lies in how to generate honeywords that are hard to be differentiated from real passwords. This security mechanism was first introduced by Juels and Rivest at CCS'13, and has been covered by hundreds of media and adopted in dozens of research domains. Existing research deals with honeywords primarily *in an ad hoc manner*, and it is challenging to develop a secure honeyword-generation method and well evaluate (attack) it. In this work, we tackle this problem *in a principled approach*. We first propose four theoretic models for characterizing the attacker $\mathcal{A}$'s best distinguishing strategies, with each model based on a different combination of information available to $\mathcal{A}$ (e.g., public datasets, the victim's personal information and registration order). These theories guide us to design effective experiments with real-world password datasets to evaluate the goodness (flatness) of a given honeyword-generation method.**

Armed with the four best attacking theories, we develop the corresponding honeyword-generation method for each type of attackers, by using various representative probabilistic password guessing models. Through a series of exploratory investigations, we show the use of these password models is not straightforward, but requires creative and significant efforts. Both empirical experiments and user-study results demonstrate that our methods significantly outperform prior art. Besides, we manage to resolve several previously unexplored challenges that arise in the practical deployment of a honeyword method. We believe this work pushes the honeyword research towards statistical rigor.

I. Introduction

Password-based authentication remains the most widely-used mechanism for authenticating users in computer systems since its advent in the 1970s. Ample of studies have revealed its security issues (e.g., guessing [14], reuse [56] and key-logging [37]) and usability problems (e.g., creation [42], memorization [26], typing [22]), various alternative authentication methods (e.g., graphical passwords [9], multi-factor authentication [36] and behavior biometrics [46]) have also been proposed. However, passwords stubbornly survive and are proliferating with almost every new web service. Gradually, a consensus is being reached in both research [15], [16], [48] and industry [3], [17], [49] that password-based authentication is likely to keep its place in the foreseeable future.

In password-based authentication systems, the server needs to maintain a sensitive password file of all users. This file provides attackers/insiders with a rich target for compromise. These years we seem to get accustomed to catastrophic password data breaches from high-profile sites (e.g., 3 billion Yahoo leak [2] and 68 million Dropbox leak [33]). Once this file is somehow obtained by the attacker $\mathcal{A}$, users' passwords are subject to offline guessing in which $\mathcal{A}$ can employ dedicated password-cracking hardware like GPU [29] and even cloud services like Amazon EC2 [8]. To address this issue, the research community has given much attention to how to store this file securely [7], [39] and why developers get password storage wrong [45], [51], and nice progress has also been made on how to measure [28], [43] and increase [10], [12] the offline guessing attacker's workload.

Relatively little attention has been given to how to timely detect the password-file leakage. It is a rare piece of good news in password research that users do tend to change their passwords when notified about password breaches [41]. However, without a timely detection mechanism, responsive countermeasures are impossible. Unsurprisingly, hundreds of popular web services have recently suffered large-scale password leaks, and most of them (e.g., Yahoo [2], MyFitnessPal [13], LiveJournal [23], Dropbox [33] and MyHeritage [38]) ask users to change passwords 1$\sim$8 *years* after the leaks originally occurred. This provides attackers enough time to crack/exploit user passwords, making the question of how to timely detect password-file compromise increasingly important.

A promising approach, named honeywords, to achieving timely password breach detection was first proposed by Juels and Rivest at CCS'13 [35]. Honeywords are decoy passwords generated for each user account, and they are stored together with the user's real password. The index of the real passwords is stored in another server of minimalist design (called honey-checker). To successfully log in, the attacker $\mathcal{A}$ has to tell the real password apart from a set of k-1 honeywords (e.g., k=20 as recommended [35]). Login with a honeyword signals a password-file leakage. The key issue lies in, when given a user account, *how to generate a set of honeywords that cannot be easily distinguished from the real password.*

Juels and Rivest [35] divided honeyword-generation methods into two categories: legacy user-interface (UI) based ones and modified-UI based ones. In legacy-UI based methods, there is no change at the user side and usability is maintained; in the modified-UI based methods, the user needs to change behavior. Because the cost of "requiring users to change behavior" is generally highly expensive [16], legacy-UI based methods are much more promising. In addition, the generation of *perfect* honeywords for modified-UI is straightforward (see Sec. 4.2 of [35]). Hence, in this work we mainly focus on legacy-UI based honeyword-generation methods.

A. Design challenges

Juels and Rivest [35] classify the legacy-UI based methods into two categories (i.e., chaffing-by-tweaking and chaffing-with-a-password-model), and three of their four primary legacy-UI methods belong to "chaffing-by-tweaking". As shown by Wang et al. [53], these four methods are all highly vulnerable. In a passing comment (see Sec. 4.1.2 of [35]), Juels and Rivest [35] do mention the possibility of using a password model to build honeywords. However, *the use of password models looks deceptively simple, but actually it is rather challenging. The following explains why.*

Firstly, it is virtually impossible to employ a password model to generate honeywords with the same probability as the user's password. User passwords are revealed well following the Zipf-like distribution [52], and this finding has been corroborated by evidence from 70 million Yahoo passwords [12]. Therefore, it is *inherently impossible* to generate enough candidate honeywords (at least 10^3 to mitigate denial-of-service attacks) that are equally probable with a relatively popular real password. This inequality gives chances to $\mathcal{A}$ to distinguish real passwords by using probabilistic approaches.

Secondly, each of the state-of-the-art password models has its own, inherent weaknesses. As briefly mentioned in [53] and in-depth investigated in this work, the PCFG-based model [56], [58] underestimates the probability of interleaving passwords (e.g., `1a2b3c4d` and `1qa2ws3ed`); the Markov-based model [40] underestimates long but meaningful passwords (e.g., `password123` and `110120130`); the List-based model (see Sec. II-D) underestimates all passwords that do not appear in the given password list (e.g., the 3 billion Yahoo list [2]), while every password list is of limited space and each service has its unique password distribution (see Fig. 3 of [56]). Such weaknesses make it improper to always use a single password model to generate honeywords, but *when* and *how* to integrate these password models to overcome the identified weaknesses has not been systematically explored.

Thirdly, the attacker $\mathcal{A}$ is powerful (yet realistic). Following the Kerckhoffs's principle, it is natural to assume that $\mathcal{A}$ knows which password model is used by the server to generate honeywords. As hundreds of sites have leaked their passwords (see [1]), and even many sites have leaked their passwords more than once (e.g., Yahoo [2], Phpbb, Ubuntu and Anthem [47]), it is also realistic to assume that $\mathcal{A}$ knows some information about the password distribution of the target service. In addition, $\mathcal{A}$ may exploit not only users' behavior of selecting popular passwords but also the victims' personally identifiable information (PII). In reality, a large fraction of users build passwords using their own PII (e.g., 36.95%$\sim$51.43% [53]), while a user's PII can often be easily learned from social networks [20] and unending data breaches [4], [27], [47]. For instance, in April 2021, the personal data of 533 million Facebook users was made freely available [34], such as name, birthday, location, phone # and email; in June 2021, personal data of 700 million LinkedIn users was sold online for $5,000 [44], including name, email, location, phone #, gender, etc.

Moreover, the registration order of users is useful for $\mathcal{A}$. This piece of info is often explicitly stored in the leaked password file (e.g., Forbes, QNB and Tianya) or implicitly reflected by the monotonically increasing user registration number. Even if it is unavailable from the leaked password file, it can often be crawled from user profiles in some applications (e.g., social/programmer forums and discussion boards), or it can be largely determined by the time when the user first participates in discussions, posts questions/answers, etc. We will show that this capability is especially useful for $\mathcal{A}$ against adaptive password-model based honeyword methods, of which the training set keeps updating as new user registers.

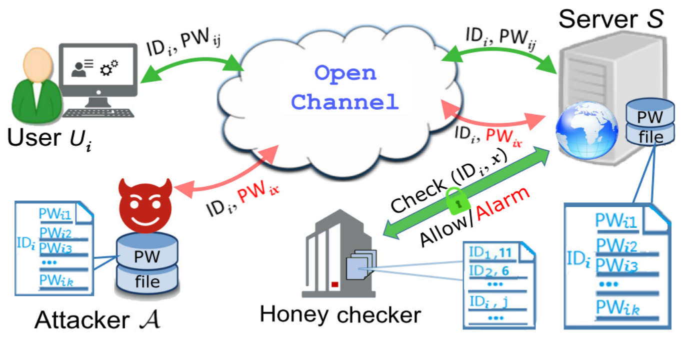
B. Related work

In 2015, Chakraborty and Mondal [21] pointed out that all of Juels-Rivest's honeyword methods [35] are random-replacement based, and thus are inherently unable to resist semantic-aware attackers. They provided *some* typical counter-example passwords (e.g., `bond007` and `john1981`) to show this. Further, they suggested a new, heuristic modified-UI honeyword method. At ACSAC'15, Almeshekah et al. [7] pointed out that the honeyword mechanism still cannot completely eliminate offline password guessing, and proposed the ErsatzPasswords scheme that employs a machine-dependent function to store passwords. Though this makes offline password guessing impossible, scalability issues arise.

In 2016, Erguler [24] also used some typical counter-example passwords to "give some remarks" about the insecurity of Juels-Rivest's four methods [35]. Since the key goal of a honeyword method is to generate honeywords indistinguishable from the user's real password, Erguler presented a new heuristic method (called "Honeyindex") that uses passwords of other users in the system as honeywords. However, the evaluation of Honeyindex is still in an ad hoc manner. Unsurprisingly, as shown in Appendix B, "Honeyindex" [24] has critical security and deployment issues.

At NDSS'18, Wang et al. [53] used heuristic experiments to reveal that the four honeyword methods by Juels-Rivest [35] all fail to achieve the claimed security in a large part: When allowed one guess, $\mathcal{A}$'s success rate can be 29.29%$\sim$32.62%, but not the expected 5%. To find potential countermeasures, they preliminarily show that existing password models (e.g., Markov [40] and PCFG [58]) each has its own inherent defects and *cannot* be readily used to generate honeywords. Accordingly, they propose to combine different password models together to overcome the defects in each individual model. However, all their experiments/proposals are still ad hoc: Whether (and when) they are optimal is unknown. Wang et al. only briefly introduced one honeyword-generation method under trawling attackers, and as shown in Sec. IV, their proposal is *not* optimal for their intended type of attackers, but desirable for another type of attackers that is not considered in [53]. Besides, they did not provide any human-attacker evaluation for their honeyword-generation method.

At 2019, Akshima et al. [6] used heuristic arguments to point out that the primary honeyword methods by Juels-Rivest [35] and Chakraborty-Mondal [21] all fail to achieve the claimed security. Further, they proposed three ad hoc honeyword-generation methods, two for legacy-UI and one for modified-UI. We show in Appendix B that their two legacy-UI methods are still subject to critical security issues.



Info of U_i	ID 4ever@foo.com;PW ******;Name john smith;Birthday 01171981
CCS'13[35] (k=14)	gracie52 tiger01 qwerty65 prince54 tiger81 gracie98 prince15 prince71 qwerty93 tiger66 qwerty10 gracie39 qwerty16 prince78
[this work] (k=14)	jsmith117 prince00 loveu123 qwertyu js128821 bond007 a123456 trustno1 rcv_11n1nj tiger81 letmein newy0rk 1989y2002r jan1981

Fig. 1. Password (PW) authentication with honeywords. For better illustration, here passwords are shown in plain-text, while in reality they are stored in salted hash. The bottom of the figure shows some personal info about the victim U_i, and exemplifies two sets of 13(=k-1) honeywords generated for U_i's password "tiger81" by two different methods: one by the hybrid method [35] and one by our TarList method (see Sec. IV).

Note that, the honeyword system is essentially a bit similar to distributed password storage (e.g., [5], [18]) that cryptographically splits passwords across two or more servers. While the former involves relatively few changes to the server side and no changes to the client side, the latter necessitates substantial changes to both sides. In addition, memory-hard functions (e.g., [10], [11]), which slow down (but cannot eliminate) password guessing, are recommended to pre-process passwords/honeywords before storage [12], [53].

In all, most prior art [6], [21], [24], [35] on honeywords mainly employs *an ad hoc approach* to design and evaluate new/existing methods. Particularly, little progress has been made towards the key question of *how best to generate and evaluate honeywords* when various types of info and varied password models are available to $\mathcal{A}$. What's more, none of the existing honeyword proposals (including [53]) have considered an attacker with user-registration order and/or the victim's PII.

C. Our contributions

Based on prior art [6], [21], [24], [35], here we take a principled approach to honeyword research. We first rigorously address the problem of how best to attack a given honeyword method under varied kinds of capabilities available to an attacker (i.e., understanding the "sword"), and then forge the "shield"—design the corresponding honeyword method based on leading password models. Our underlying rationale is that, only when one knows what's $\mathcal{A}$'s best attacking strategy, one can figure out how to design the most effective countermeasures. In all, we make the following key contributions:

- **Attacking theories**. To characterize the attackers' best strategies, we, *for the first time*, propose a series of theoretic models based on varied kinds of capabilities available to an attacker. Particularly, we are the *first* to consider the realistic attackers that exploit each victim's personal information and know the order of user registration. These models enable us to design effective experiments with real-world datasets to evaluate the goodness (flatness) of a generation method, answering the open question left by Juels and Rivest [35].
- **Generation methods**. We develop four novel and efficient honeyword-generation methods based on various existing probabilistic password-cracking models (e.g., Markov [40], PCFG [58] and TarGuess PCFG [56]). The use of these password models *requires significant, novel and creative efforts*, and we show this by a series of exploratory investigations. Our constructions not only resolve Juels-Rivest's question [35], but also give a way to retool cracking models to build flat honeywords, enabling future improvements of cracking models to be easily incorporated into honeyword methods.
- **An intensive evaluation**. We implement our new methods and show their effectiveness by performing extensive experiments under four kinds of attackers ($\mathcal{A}_1$~$\mathcal{A}_4$), each based on a different combination of info available to $\mathcal{A}$: public datasets, user PII and registration order. Our experiments build on 11 large-scale datasets, including 105.44 million real-world passwords. To see how they perform under semantic-aware humans, we further conduct a user study of 11 trained human attackers. Results indicate that our methods can survive both automated and human attackers.
- **Some insights**. We obtain a number of insights, some expected and some surprising, from our theories and experiments. Our attacking theories show that the "chaffing-by-tweaking" category of methods is inherently problematic, and all such methods are far from flat (distinguishable). This is opposed to the common belief in [35], is corroborated by the empirical results in [53] and necessitates the design of password-model based methods. As expected, password models can be used to build flat honeywords, but somewhat surprisingly, the adaptive List model (not the expected PCFG or Markov [40]) can generate nearly flat honeywords under the basic attacker who is with only public datasets.

II. Preliminaries

A. System model

As shown in Fig. 1, four entities are involved in the honeyword system: a user U_i, an authentication server S, a honeychecker, and the attacker $\mathcal{A}$. User U_i first creates an account $(\text{ID}_i, \text{PW}_i)$ at the server S. Some PII may also be provided to S, and this enables S to employ PII-aware honeyword methods. Besides the normal procedures for user registration, S carries out a command $\mathsf{Gen}(k; \text{PW}_i)$: S generates a list of k-1 distinct, decoy passwords (called *honeywords*) to store along with U_i's real password PW_i, where k=20 as suggested in [35]. PW_i and its k-1 honeywords are called k *sweetwords*.

Generally, honeyword-generation methods can be classified into two broad categories: password-model based (see Sec. IV) and random replacement based (i.e., chaffing-by-tweaking in [35]). Generally, random replacement based methods are also real-password related: They generate honeywords explicitly relating to the real password (e.g., tweaking-tail [35]); password-model based methods are real-password *un*related, i.e., they generate honeywords independent of the real password (e.g., Honeyindex [24] and all the four new methods in this work).

Without loss of generality, we use Juels and Rivest's first method [35], i.e. "Tweaking by tail" [35], as a representative of *non*-password-model based methods. This method "tweaks" the selected character positions of the real password PW_i to generate the $k-1$ honeywords. Let t (e.g., $t = 2$ or 3) denote the desired number of positions to tweak. Each character in the last t positions of PW_i is substituted by a randomly-selected character of the same type: A digit is substituted by a digit, a letter by a letter, and a symbol by a symbol. For example, if PW_i is `loveu1`, k=4 and t=2, then the sweetword list SW_i for U_i might be {`lovea0`, `lovex7`, `lovee0`, `lovey3`}.

B. Security model

Honeyword distinguishing attacker. The essential security goal of any honeyword method is, when given user U_i's account, to generate a set of k-1 honeywords such that they are *in*distinguishable from U_i's real password PW_i. This goal is defined against the honeyword distinguishing attacker $\mathcal{A}$ as shown in Fig. 1, who has obtained the sweetword file, offline guessed all the users' sweetwords and employed S as an online querying oracle. $\mathcal{A}$'s honeyword online querying attempts will be detected by the honeychecker, if $\mathcal{A}$ uses a honeyword to log in. If the number of honeyword login exceeds the per-user threshold $\mathcal{T}_1$ (e.g., 3), $\mathcal{A}$ will raise the alarm on U_i's account. $\mathcal{A}$ will also cause the system-wide alarm to be raised if $\mathcal{A}$'s honeyword login attempts exceed the system-wide threshold $\mathcal{T}_2$ (e.g., 10^4). Thus, to avoid being detected, $\mathcal{A}$ shall try honeywords as few as possible. Note that, the exact values of $\mathcal{T}_1$ and $\mathcal{T}_2$ depend on the target system's risk analysis results, and are out of our scope. This explains why they have not been discussed in the existing literature. Still, since the system has to balance honeyword-distinguishing attacks and denial-of-service (DoS) attacks, $\mathcal{T}_2$ should not be too small or too large, and without loss of generality, we set $\mathcal{T}_2$=10^4 as with [53].

Attacker capabilities. As shown in Table I, we assume that $\mathcal{A}$ has somehow already got access to the server S's password hash file, and knows all public info such as leaked password lists, password policy and the honeyword method used by S to generate honeywords. As hundreds of sites have leaked their passwords (see [1]), $\mathcal{A}$ may also know some info about the password distribution of the target system. This kind of attacker (i.e., type-$\mathcal{A}_1$) is the basic attacker, and it has been (implicitly) made in existing studies [6], [18], [24], [35]. To make our attacking models more realistic, we also investigate the scenario where some sweetwords are unknown to $\mathcal{A}$.

As users love to build passwords using their own PII, a practical method should not overlook this information. In addition, users' registration order is also useful for $\mathcal{A}$. This info is especially useful for $\mathcal{A}$ against adaptive password-model based honeyword-generation methods. In such adaptive methods, the underlying password-model keeps updating with newly registered passwords, and newly generated honeywords will only depend on existing passwords but not the future passwords (similar to Honeyindex [24] and Akshima et al.'s methods [6], which are analyzed in Appendix B). Therefore, $\mathcal{A}$ can attack in the same order as the user registration order. As mentioned in Sec. I, this piece of info is often not considered sensitive and can be obtained/inferred in a number of ways.

TABLE I
ATTACKER CAPABILITIES CONSIDERED IN THIS WORK.

Attacker type		PW file	Public info[1]	Personally identifiable info[2]	User registration order	Existing literature
Distinguishing attacker	$\mathcal{A}_1$	✓	✓			[6], [24], [35], [53]
	$\mathcal{A}_2$	✓	✓	✓		[53][3]
	$\mathcal{A}_3$	✓	✓		✓	None
	$\mathcal{A}_4$	✓	✓	✓	✓	None

[1] Typical public info includes the various leaked password lists, password policy and all the cryptographic algorithms (e.g., hash methods and the honeyword-generation methods).
[2] Such as name, birthday, gender, email, education and hobbies.
[3] In [53], PII is only considered for attacking, but not for defense.

Other attackers. As discussed in [6], [24], [35], [53], other threats against honeywords, such as multi-system intersection attacks, DoS attacks and honeychecker-related attacks, are also practical concerns. Fortunately, most of them can be well mitigated. For example, multi-system intersection attacks arise because users tend to reuse passwords across different services, and they can be thwarted by cryptographic means [57]. To resist DoS attacks (that deliberately login with honeywords to raise alarms), we can focus on producing flat honeywords in the generation phase, and the server S can take proper measures (without sacrificing too much security/usability) in the authentication phase. For example, S can employ stricter rate-limiting policies and Captcha schemes to thwart malicious login attempts, and set customized alarm policies to give more weight to strong honeywords than weak ones (as it would be more difficult to guess strong honeywords correctly [6]). Also note that flatter password-model based honeyword methods might be easier to DoS attacks, because popular passwords are now more likely to be selected as honeywords. Thus, S can further employ blocklists and password strength meters (PSM, like fuzzyPSM [54] and Zxcvbn [59] as suggested in [28]) to reduce the use of weak passwords during user registration, and in this case, weak honeywords shall similarly be blocked.

C. Evaluation metrics

This work adopts the two evaluation metrics proposed in [53] to measure the *advantages* of a distinguishing attacker, or equally the *goodness* of a honeyword method.

Flatness graph plots the chance y of finding the real password by making x login attempts per user, where y ∈[0, 1] and $x{\le}k$ (actually, $x{\le}\mathcal{T}_1$). This metric measures the *average-case* performance of a honeyword method. The ϵ-flat metric introduced in [35] is just the first data point (x=1, ϵ=$y|_{x=1}$) on the flatness graph, i.e., the ϵ-flat metric is incorporated.

Success-number graph plots the number y of successfully identified real passwords, when the attacker $\mathcal{A}$ has made a total of x honeyword login attempts, where $x{\le}\mathcal{T}_2$. To find more real passwords, the best strategy for $\mathcal{A}$ is to first try these most probable passwords. Thus, this metric measures the *worst-case* performance of a method.

D. Probabilistic password models

We introduce six representative probabilistic password models our new methods build on: PCFG [58], Markov [40], List

[53], and their corresponding targeted versions converted by using the PII type-based tags [56]. They all require a training set. We do not consider the neural-network-based model, because it is ineffective when $\mathcal{A}$'s guess number is small (e.g., $\leq\mathcal{T}_2$) [43] and thus unsuitable for honeyword settings.

PCFG. This model was first introduced by Weir et al. [58], and it has been established to be one of state-of-the-art password cracking algorithms by recent research (e.g., [40], [42], [56]). This model treats passwords as a combination of segments. For example, "`wanglei@123`" is divided into the L segment "wanglei", S segment "@" and D segment "123", and its base structure is $\mathsf{L}_7\mathsf{S}_1\mathsf{D}_3$. The probability of "`wanglei@123`" $\Pr(\texttt{wanglei@123})$ is the product of $\Pr(\mathsf{L}_7\mathsf{S}_1\mathsf{D}_3)$, $\Pr(\mathsf{L}_7 \rightarrow \texttt{wanglei})$, $\Pr(\mathsf{S}_1 \rightarrow \texttt{@})$ and $\Pr(\mathsf{D}_3 \rightarrow \texttt{123})$.

Markov. Unlike the PCFG-based model, there is a parameter determining the Markov-based model [40]—the order of the Markov chains. A Markov chain of order d, where d is a positive integer, is a process with a Markov assumption:

$$\Pr(c_i|c_1c_2\cdots c_{i-2}c_{i-1}) = \Pr(c_i|c_{i-d}\cdots c_{i-1}).$$
$$=\frac{\texttt{Count}(c_{i-d}\cdots c_{i-1}c_i)}{\texttt{Count}(c_{i-d}\cdots c_{i-1}c_{i-1})}.$$

where $\texttt{Count}(c_{i-d},\cdots,c_{i-1},c_i)$ denotes the number of occurrences of the string $c_{i-d}\cdots c_{i-1}c_i$ in the training set. That is, the probability of the next character in a string is based on a prefix of length d. Then, the probability of a string s=$c_1c_2\cdots c_n$ is:

$$\Pr(s) = \Pr(c_1)\Pr(c_2|c_1)\cdots\Pr(c_n|c_{n-1}c_{i-2}\cdots c_1)$$
$$= \prod_{i=1}^{n}\Pr(c_i|c_{i-1}\cdots c_{i-d}).$$

List is a simple yet useful model: $\forall s \in \mathcal{D}$, $P_{\mathcal{D}}(s) = \frac{\texttt{Count}(s)}{|\mathcal{D}|}$, where $\mathcal{D}$ is a multi-set (e.g., a leaked password dataset) and $\texttt{Count}(s)$ is the occurrences of password s in $\mathcal{D}$.

TarPCFG. This model was first proposed in [56] and also called TarGuess-I. Besides the L, D, S tags originally defined in PCFG [58], TarPCFG defines a series of new type-based PII tags (e.g., $\mathsf{N}_1{\sim}\mathsf{N}_7$ and $\mathsf{B}_1{\sim}\mathsf{B}_{10}$). For a *type-based* PII tag, its subscript number denotes *a particular sub-type* of one kind of PII usages but not the *length* matched, contrary to the L, D, S tags. For instance, N stands for all kinds of name usages, where N_1 for full name (e.g., `wang lei`) and N_2 for family name (e.g., `wang`); B stands for all kinds of birthday usages, and B_1 for full birthday in YMD format, etc. Each PII tag can then be operated in the same way with L/D/S tags. TarPCFG outperforms PCFG by 412%$\sim$740% within 100 guesses.

TarMarkov. As shown in [56], to convert a traditional Markov model into a PII-enriched Markov model, one only needs to include the type-based PII tags $\{\mathsf{N}_1,\ldots,\mathsf{N}_7;\mathsf{B}_1,\ldots,\mathsf{B}_{10};\ldots\}$ into the alphabet Σ (e.g., $\Sigma = \{$95 printable ASCII characters$\}$ in [40]) of the Markov n-gram model, and all operations for these PII tags are the same with the atomic characters in Σ.

TarList. As the List model can be essentially seen as a PCFG *without* the L, D and S tags, it can be similarly converted into a targeted model with that of PCFG.

TABLE II
BASIC INFO ABOUT OUR 10 PASSWORD DATASETS.†

Dataset	Web service	Language	When leaked	Total PWs	With PII
Tianya	Social forum	Chinese	Dec., 2011	30,901,241	
Dodonew	E-commerce	Chinese	Dec., 2011	16,258,891	
CSDN	Programmer	Chinese	Dec., 2011	6,428,277	
Mango	E-commerce	Chinese	July, 2015	1,074,742	
Rockyou	Social forum	English	Dec., 2009	32,581,870	
000webhost	Web hosting	English	Oct., 2015	15,251,073	
Yahoo	Web portal	English	July, 2012	442,834	
12306	Train ticketing	Chinese	Dec., 2014	129,303	✓
ClixSense	Paid task platform	English	Sep., 2016	2,222,045	✓
Rootkit	Hacker forum	English	Feb., 2011	69,418	✓
QNB*	E-bank	English	April, 2016	79,580	✓

†PW stands for password, PII for personally identifiable information.
*QNB passwords are from e-Bank and used as high-value targets.

TABLE III
BASIC INFORMATION ABOUT OUR PII DATASETS.†

Dataset	Language	Items num	Types of PII useful for this work
Hotel	Chinese	20,051,426	Name, Birthday, Phone, NID*
51job	Chinese	2,327,571	Email, Name, Birthday, Phone
12306	Chinese	129,303	Email, User name, Name, Birthday, Phone, NID
ClixSense	English	2,222,045	Email, User name, Name, Birthday
Rootkit	English	79,580	Email, User name, Name, Birthday
QNB	English	77,799	Email, User name, Name, Birthday, Phone, NID

† NID=National identification number, e.g., social security number.

Generating honeywords. Since probabilistic password models can produce a set of passwords (usually called guesses) with probabilities, honeywords can be generated by uniformly sampling from this probability space. For the List model, we directly sample from this set of guesses; For other more complex models, it is computationally prohibitive to explicitly generate this set of guesses and then directly sample from it. Instead, we sample from the interim probabilistic products by using inverse CDF. Take PCFG [58] for an example. We first use the inverse CDF to (uniformly) sample from $\Pr(\mathcal{S}{\rightarrow}\cdot)$ to obtain the base structure $\mathsf{L}_7\mathsf{D}_3$, and then similarly obtain segments `wanglei` from $\Pr(\mathsf{L}_7{\rightarrow}\cdot)$ and `123` from $\Pr(\mathsf{D}_3{\rightarrow}\cdot)$, respectively. For hybrid models, we first determine which password model to be used according to their weights, and then perform inverse CDF sampling on the selected model. For targeted models (e.g., TarPCFG [56]), we first similarly obtain samples which contain PII tags, and then substitute the tags with user PII. For example, for user "John Smith", sampling from TarPCFG may first produce the interim item N_2`123`. Then, by replacing the family name tag N_2 with his family name `Smith`, we obtain the final honeyword `Smith123`. For concrete examples of generated honeywords, see the bottom of Fig. 1. Our constructions in Sec. IV follow this basic idea, but further address a number of challenges.

E. Our datasets and ethics consideration

We evaluate the existing honeyword methods and our proposed ones based on 11 large real password datasets (see Table II and Table III), a total of 105.44 million passwords. Our password datasets include four from English sites and five from Chinese sites. For better comparison, these datasets except Mango are the same with [53].

Particularly, four of our password datasets (i.e., 12306, ClixSense, Rootkit and QNB) are associated with various kinds of PII. To enable extensive targeted attacks, we obtain nine PII-associated password datasets (see Table IV) by matching the *non*-PII-associated password datasets with these

TABLE IV

BASIC INFO OF OUR NINE PII-ASSOCIATED PASSWORD DATASETS THAT ARE CONSTRUCTED BY MATCHING EMAIL.

Dataset name	PII-Dodonew	PII-12306	PII-CSDN	PII-Rootkit	PII-000webhost	PII-ClixSense	PII-Yahoo	PII-QNB
Size	161,517	129,303	77,216	69,330	153,390	2,222,045	16,307	77,799

PII-associated ones through email. As the canonical dataset Rockyou only consists of passwords (with no user names or emails), it will not be used when evaluating targeted threats. For more information about our datasets, see Appendix A.

For ethical considerations, we only present the aggregated statistical information and protect each individual account as confidential. All our datasets are stored and processed on computers not linked to the Internet. In addition, recovered password hashes are deleted once the data analysis is completed, though it is likely that attackers would have already cracked most of them [8], [29]. Our measures can secure the recovered passwords just in case when we have cracked some hashes that attackers have not, ensuring that no new risks is brought to victim users. While attackers may exploit these datasets for misconduct, our use is both beneficial for the academic community to understand honeyword choices and for security administrators to timely detect password leaks. As our datasets are all publicly available from the Internet, this work is reproducible.

III. OUR ATTACKING THEORIES

We now propose a series of theoretic models for characterizing the attacker $\mathcal{A}$'s best attack strategies in telling apart the real password from a set of honeywords. Each of these models is based on varied kinds of information available to $\mathcal{A}$ (Table I). Particularly, we are the first to consider quite realistic attackers who know user registration order. These models guide us to design effective experiments with real datasets to evaluate a given honeyword method.

A. Theoretical attacking models

As implied in Sec. II-B, there are two main goals of the distinguishing attacker $\mathcal{A}$: (1) Global goal—identifying *real passwords* from a given password file F composed of n sweetword lists $\{\mathrm{SW}_1, \mathrm{SW}_2, \ldots, \mathrm{SW}_n\}$, where SW_i=$(sw_{i,1}, \ldots, sw_{i,k})$, $1 \leq i \leq n$; and (2) Local goal—identifying *the real password* from a given sweetword list SW_i of user U_i. Since $\mathcal{A}$ can at most make $\mathcal{T}_2$ overall failed attempts for the system and $\mathcal{T}_1$ failed attempts per user, to achieve each goal she needs to try these most probable sweetwords first.

Basic idea. We first show that $\mathcal{A}$'s two goals can be best achieved by using the same attacking strategy. Next, by analyzing and exploiting various properties of the two categories of honeyword methods, we formulate what a basic attacker $\mathcal{A}_1$'s optimal strategy is. Finally, we extend the best attacking strategy for $\mathcal{A}_1$ to the other three types of attackers $\mathcal{A}_2$~$\mathcal{A}_4$.

Theorem 1: Let $\mathsf{pw}_{i,j}$ $(1 \leq j \leq k)$ denote the event that U_i selects $sw_{i,j}$ as her real password, and $\mathsf{hw}_{i,t}$ denote the event that $sw_{i,t}$ is produced as a honeyword for U_i. We have

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i) = \frac{\Pr(\mathsf{pw}_{i,j}) \prod_{l\neq j} \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j})}{\sum_{t=1}^{k} \Pr(\mathsf{pw}_{i,t}) \prod_{l\neq t} \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t})}, \quad (1)$$

under the assumption that $\mathsf{hw}_{i,1}, \ldots, \mathsf{hw}_{i,j-1}, \mathsf{hw}_{i,j+1}, \ldots, \mathsf{hw}_{i,k}$ are *mutually* independent under the event $\mathsf{pw}_{i,j}$.[1]

The detailed proof can be found in Appendix C. This theorem indicates that $\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i)$ can be computed if $\Pr(\mathsf{pw}_{i,j})$ and $\Pr(\mathsf{hw}_{i,t}|\mathsf{pw}_{i,j})$ are known. Fortunately, $\Pr(\mathsf{pw}_{i,j})$ can be obtained by using various password models (e.g., the List model—directly from a leaked password dataset), and $\Pr(\mathsf{hw}_{i,t}|\mathsf{pw}_{i,j})$ can be obtained by analyzing the properties of a given honeyword method.

Theorem 2: Let F denote the event that the file F is produced as the password-file for all users, and the other definitions comply with those in Theorem 1. We have

$$\Pr(\mathsf{pw}_{i,j}|F) = \Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i), \quad (2)$$

under the assumptions that users independently create passwords, and the assumptions of Theorem 1.

The detailed proof can be found in Appendix C. This theorem indicates that, finding the most probable password in $\mathrm{SW}_1, \ldots, \mathrm{SW}_n$ can be reduced into first finding the most probable password *within each sweetword list* and then ranking these candidate passwords. In this light, attacker $\mathcal{A}$'s two goals can be essentially achieved using the same attacking strategy.

We now summarize four properties that a honeyword method may have. These properties can be used to classify existing honeyword methods into two cases, and then we *simplify* the computation of Eq. 1 for each case. The domain of sweetwords is by default $\mathcal{D}_{pw}$, the entire password space. Let $T(x)$ denote the sweetword space of x, that is, the set of sweetwords obtainable from password x. We define:

P1: $\forall sw_1 \forall sw_2,\ sw_1 \in T(sw_2) \Longrightarrow T(sw_1) = T(sw_2)$.
This property states that any sweetword can generate any other sweetword in a sweetword list. This is a desirable property to achieve, because suppose $sw_{i,j} \in \mathrm{SW}_i$ cannot generate some sweetwords that appear in the list SW_i, then it is certain that $sw_{i,j}$ will *not* be the real password. Otherwise, a paradox arises. The attacker $\mathcal{A}$ can thus easily eliminate some non-real passwords like $sw_{i,j}$ without trying to log in. To formalize, this property indicates that $\forall t, T(sw_{i,t}) = T(sw_{i,1})$, and $\forall t, \forall l \neq t, \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t}) \neq 0$.

P2: $\forall sw_1 \forall sw_2, \forall sw_3 \in T(sw_1) \cap T(sw_2)$,
$\Pr(\mathsf{hw}_3|\mathsf{pw}_1) = \Pr(\mathsf{hw}_3|\mathsf{pw}_2)$.
This property states that every sweetword can be generated by any candidate password with equal probability. All honeyword-generation methods we discuss satisfy P1 and P2. For example, under $t = 2$ tweaking-tail method, the probabilities of sweetword `lovea0` conditioned on password being respectively `loveu1` and `lovee0` are the same.

[1]Note that, the assumptions in Theorem 1 comply with the fact that the events $\mathsf{hw}_{i,1}, \ldots, \mathsf{hw}_{i,j-1}, \mathsf{hw}_{i,j+1}, \ldots, \mathsf{hw}_{i,k}$ may be dependent or independent on the event $\mathsf{pw}_{i,j}$.

P3: $\forall sw_1, \forall sw_2 \forall sw_3 \in T(sw_1), \Pr(\mathsf{hw}_2|\mathsf{pw}_1) = \Pr(\mathsf{hw}_3|\mathsf{pw}_1)$. This property states that every sweetword generates all sweetwords in its sweetword space with the same probability. All non-password-model based methods we discuss here (e.g., tweaking-tail) satisfy this property.

P4: $\forall sw_1 \forall sw_2 \forall sw_3, \Pr(\mathsf{hw}_3|\mathsf{pw}_1) = \Pr(\mathsf{hw}_3|\mathsf{pw}_2)$. This property states that honeywords are unrelated with the real password; when combined with property P1, it indicates that $\forall pw, T(pw) = \mathcal{D}_{pw}$.

Case 1. For these methods (i.e., all of non-password-model based methods, like tweaking-tail [35]) that satisfy the properties P1~P3, it follows that $\forall i \forall j, \forall l \neq j, \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j}) \equiv c$, where $c > 0$ is a constant. Now, we can derive

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i) = \frac{\Pr(\mathsf{pw}_{i,j})}{\sum_{t=1}^{k} \Pr(\mathsf{pw}_{i,t})} , \quad (3)$$

where both $\Pr(\mathsf{pw}_{i,j})$ and $\Pr(\mathsf{pw}_{i,t})$ can be obtained by using various password models (e.g., List, PCFG-based [58]).

Eq. 3 fundamentally explains why the "normalized top-PW" attacking strategy in Sec. 3.2 of [53] is effective: This strategy accords with our above theory and is the optimal one against Juels-Rivest's four methods [35]. It in turn suggests that, for these honeyword methods in [35] to achieve $\frac{1}{k}$-flat (i.e., to be perfect), all k sweetwords in the list SW_i shall have an equal probability to be selected as U_i's real password PW_i. In other words, given the real password PW_i, a real-password related method needs to produce $k-1$ honeywords with equal probability to be U_i's password as PW_i a priori. This is *inherently unachievable* due to the fact that user-chosen passwords follow the Zipf's law [52]: $p_r = C \cdot r^s$- $C \cdot (r-1)^s$ $\approx C \cdot s \cdot r^{s-1}$, where p_r denotes the probability of the rth popular password, and $s\in$[0.15, 0.30] and $C\in$[0.001, 0.1] are constants. As p_r decreases sharply with r when r is small, it is difficult to find suitable honeywords for relatively popular real passwords. *This outlines the need for new design techniques beyond the real-password related ones.*

Case 2. For methods (e.g., Honeyindex [24] and all our methods) that comply with the properties P1, P2 and P4, it follows that $\forall sw_1, sw_2, sw_3 \in \mathcal{D}_{pw}$, $\Pr(\mathsf{hw}_3|\mathsf{pw}_1) = \Pr(\mathsf{hw}_3|\mathsf{pw}_2)$. For simplicity, we write $\Pr(\mathsf{hw}_3|\mathsf{pw}_1)$ as $\Pr_{\mathrm{HW}}(sw_3)$, which means that the probability of producing sw_3 as a sweetword only depends on the underlying honeyword method HW. Similarly, we use $\Pr_{\mathrm{PW}}(sw_1)$ to denote $\Pr(\mathsf{pw}_1)$: The probability of selecting the string sw_1 as a real password. Now, we can simplify Eq. 1 to be:

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i) = \frac{\frac{\Pr_{\mathrm{PW}}(sw_{i,j})}{\Pr_{\mathrm{HW}}(sw_{i,j})}}{\sum_{t=1}^{k} \frac{\Pr_{\mathrm{PW}}(sw_{i,t})}{\Pr_{\mathrm{HW}}(sw_{i,t})}} , \quad (4)$$

where both $\Pr_{\mathrm{PW}}(sw_{i,j})$ and $\Pr_{\mathrm{PW}}(sw_{i,t})$ can be obtained by using various password models (e.g., List and Markov-based [40]), and both $\Pr_{\mathrm{HW}}(sw_{i,j})$ and $\Pr_{\mathrm{HW}}(sw_{i,t})$ can be computed by the corresponding honeyword-generation method.

Eq. 4 points out how to best attack all our honeyword-generation methods, in which honeywords are independent of the user's real password. It in turn suggests that, for such methods to achieve perfect flatness, they shall satisfy $\forall pw \in \mathcal{D}_{pw}, \Pr_{\mathrm{HW}}(pw) = \Pr_{\mathrm{PW}}(pw)$. In other words, the probability distribution model (function) $\Pr_{\mathrm{HW}}(\cdot)$ of such methods shall be equal to the password distribution model $\Pr_{\mathrm{PW}}(\cdot)$ of the authentication system. It conforms to our intuition, because password and honeyword would be indistinguishable if they have the same distribution.

This in turn indicates that, for a method to be perfect, $\Pr_{\mathrm{HW}}(\cdot)$ shall be *the same* with $\Pr_{\mathrm{PW}}(\cdot)$, while the latter primarily depends on the underlying system (e.g., language, service type and password policy) and can be approximated by various probabilistic password models (e.g., List, PCFG-based [58], Markov-based [40] and TarGuess [56]). *This suggests that these password models can be potentially employed to build honeyword methods (i.e.,* $\Pr_{\mathrm{HW}}(\cdot)$*)*, and we show how to make it a reality in the next section.

B. Three extensions

In the above, we have only investigated the best attacking strategies for a distinguishing attacker $\mathcal{A}$ who is with capabilities of $\mathcal{A}_1$ (see Table I). $\mathcal{A}_1$ does not exploit user PII or user registration order. Yet, in reality a large fraction of users (i.e., 36.95%~51.43% [53]) build passwords with their own PII; in Sec. IV, we show user registration order can also be exploitable. We now provide the best attacking theory for attackers $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$, respectively.

We demonstrate that, similar as attackers of type-$\mathcal{A}_1$, the attackers of type- $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ have the same equations of best attacking theories except with one more condition X.

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{X}) = \frac{\Pr(\mathsf{pw}_{i,j}|\mathsf{X}) \prod_{l \neq j} \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j}, \mathsf{X})}{\sum_{t=1}^{k} \Pr(\mathsf{pw}_{i,t}|\mathsf{X}) \prod_{l \neq t} \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t}, \mathsf{X})}, \quad (5)$$

where the condition X is the personally identifiable information (PII) for type-$\mathcal{A}_2$ attacker, the registration order Reg for type-$\mathcal{A}_3$ attacker and (PII, Reg) for type-$\mathcal{A}_4$ attacker. In what follows, we take X=PII for a concrete example.

Theorem 3: Let $\mathsf{pw}_{i,j}$ $(1 \leq j \leq k)$ denote the event that U_i selects $sw_{i,j}$ as her password, $\mathsf{hw}_{i,j}$ denote that $sw_{i,j}$ is produced as U_i's honeyword, and PII denote U_i's PII. We have

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{PII}) = \frac{\Pr(\mathsf{pw}_{i,j}|\mathsf{PII}) \prod_{l \neq j} \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j}, \mathsf{PII})}{\sum_{t=1}^{k} \Pr(\mathsf{pw}_{i,t}|\mathsf{PII}) \prod_{l \neq t} \Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t}, \mathsf{PII})},$$

under the assumption that $\mathsf{hw}_{i,1}, \ldots, \mathsf{hw}_{i,j-1}, \mathsf{hw}_{i,j+1}, \ldots, \mathsf{hw}_{i,k}$ are *mutually* independent under the event $(\mathsf{pw}_{i,j}, \mathsf{PII})$.

The detailed proof can be found in Appendix C.

We now instantiate/simplify Eq. 5, and derive the optimal honeyword attacking strategies for type-$\mathcal{A}_2$~$\mathcal{A}_4$ attackers under these two cases of different honeyword methods.

For attackers of type-$\mathcal{A}_2$. As with a type-$\mathcal{A}_1$ attacker, there are two *cases* (see the detailed definition in Sec. III-A) to be considered for different types of honeyword methods.

Similar to Eq. 3, Eq. 6 explains that for non-password-model based methods to be secure, probabilities of all sweetwords in the same sweetword space should be approximately the same under targeted password models. This is more difficult to achieve than the trawling scenario, because the probabilities of targeted password models would change more drastically

[56]. Eq. 7, like Eq. 4, shows that for password-model based methods to be secure, the targeted distribution of honeywords should be close enough to the distribution of passwords.

$$Case1: \quad \Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{PII}) = \frac{\Pr_{\mathrm{PW}}(sw_{i,j}|\mathsf{PII})}{\sum_{t=1}^{k} \Pr_{\mathrm{PW}}(sw_{i,t}|\mathsf{PII})}. \quad (6)$$

$$Case2: \quad \Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{PII}) = \frac{\frac{\Pr_{\mathrm{PW}}(sw_{i,j}|\mathsf{PII})}{\Pr_{\mathrm{HW}}(sw_{i,j}|\mathsf{PII})}}{\sum_{t=1}^{k} \frac{\Pr_{\mathrm{PW}}(sw_{i,t}|\mathsf{PII})}{\Pr_{\mathrm{HW}}(sw_{i,t}|\mathsf{PII})}}. \quad (7)$$

For attackers of type-$\mathcal{A}_3$. As user registration order is mainly meaningful for password-model based methods, a type-$\mathcal{A}_3$ attacker is restricted to Case 2 in Sec. III-A. More specifically, for Eq. 4, the attacker $\mathcal{A}$ can improve $\Pr_{\mathrm{PW}}(sw_{i,j})$ to be $\Pr_{\mathrm{PW}}(sw_{i,j}|\mathsf{Reg})$ and $\Pr_{\mathrm{HW}}(sw_{i,t})$ to $\Pr_{\mathrm{HW}}(sw_{i,t}|\mathsf{Reg})$ by using user registration order (e.g., by adaptively updating her training set):

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{Reg}) = \frac{\frac{\Pr_{\mathrm{PW}}(sw_{i,j})}{\Pr_{\mathrm{HW}}(sw_{i,j}|\mathsf{Reg})}}{\sum_{t=1}^{k} \frac{\Pr_{\mathrm{PW}}(sw_{i,t})}{\Pr_{\mathrm{HW}}(sw_{i,t}|\mathsf{Reg})}}. \quad (8)$$

For attackers of type-$\mathcal{A}_4$. When a type-$\mathcal{A}_3$ attacker is further equipped with user PII, she can improve her advantages by combining Eqs. 7 and 8 to get:

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{PII}, \mathsf{Reg}) = \frac{\frac{\Pr_{\mathrm{PW}}(sw_{i,j}|\mathsf{PII})}{\Pr_{\mathrm{HW}}(sw_{i,j}|\mathsf{PII},\mathsf{Reg})}}{\sum_{t=1}^{k} \frac{\Pr_{\mathrm{PW}}(sw_{i,t}|\mathsf{PII})}{\Pr_{\mathrm{HW}}(sw_{i,t}|\mathsf{PII},\mathsf{Reg})}}. \quad (9)$$

C. Applications of our attacking theories

We now show how to apply the above attacking theories to design more effective attacks. Without loss of generality, here we take the tweaking-tail method in [35] as the target method. First of all, since Theorems 1 and 2 are general (according to their definitions), so they can be readily applied to any method. The next step is to choose the right attacking theories by analyzing the properties of the method under study. Since the tweaking-tail method satisfies the Property 1∼3, for a type-$\mathcal{A}_1$ attacker, Eq. 3 is applicable; for a type-$\mathcal{A}_2$ attacker, Eq. 6 is applicable. Since U_i's honeywords generated by the tweaking-tail method only relate to U_i's real password PW_i, the registration order will be useless. Thus, in this case, type-$\mathcal{A}_3$ and $\mathcal{A}_4$ attackers will not be more effective than type-$\mathcal{A}_1$ and $\mathcal{A}_2$ ones, respectively. In all, mainly Eqs. 3 and 6 are helpful for attacking the tweaking-tail method.

Essentially, Wang et al.'s empirical evaluations [53] of Juels-Rivest's four methods [35] can be seen as some applications of our attacking theories Eqs. 3 and 6, while their empirical evaluations of two password-model based methods (i.e., PCFG and Markov) can be seen as the applications of our attacking theories Eq. 4. More specially, their "Norm top-PW" attacks (see Fig. 7 of [53]) against Juels-Rivest's four methods [35] are instantiations of Eq. 3 for the type-$\mathcal{A}_1$ attacker, and Eqs. 6 for the type-$\mathcal{A}_2$ attacker; their "Norm PW-model" attacks (see Fig. 10 of [53]) against password-model based methods are instantiations of Eq. 4. This explains why Wang et al.'s attacks are effective, and resolves their open question [53]: "Is our attacking strategy optimal?"

Wang et al. [53] show that, if the attacker $\mathcal{A}_2$ exploits the victim's PII but the honeyword generation method does not consider user PII, $\mathcal{A}_2$ can indeed improve her chance. Also take the Tweaking-tail method as an example. As revealed in Fig. 7(a) of [53], $\mathcal{A}_2$ can guess 47.1%∼61.3% more real passwords when $\mathcal{T}_2$=10^4, and achieve 40.9%∼51.6% more success rates in terms of the ϵ-flatness metric (see the point (x=1, y=0.5) in Fig. 7(d) of [53]). What's most disturbing is that, against every method, PII-enriched attackers now can attain over 49.5% of success rates in distinguishing the real password from 19 honeywords with only *one* guess (i.e., being 0.495^{+}-flat), while the desirable, optimal security is 0.05-flat.

In the above we assume that all hashed sweetwords can be cracked and known to $\mathcal{A}$, yet in reality there might be a portion of sweetwords that are difficult to be recovered. Still, this new assumption does not change our optimal attacking strategies in terms of flatness: Now the attacker only needs to apply our strategy to these cracked sweetwords. This is essentially the "nut" strategy in [35]: The attacker is more likely to crack user-chosen passwords, but not hard honeywords ("nuts").

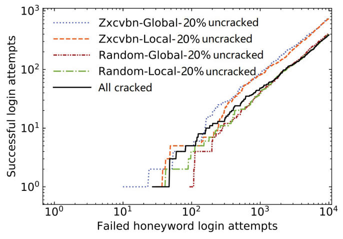


Fig. 2. Success-number graph of the honeyword method $\frac{1}{3}$List $+\frac{1}{3}$Markov $+\frac{1}{3}$PCFG (see Sec. IV). Trained on Dodonew-tr, tested on Dodonew-ts. Uncracked sweetwords are selected in four ways: randomly or deemed strong by Zxcvbn [59], per account (Local) or among all sweetwords (Global).

Comparatively, it is challenging to derive the optimal strategy in terms of success number under the new assumption, but we empirically show that simple approximations can work very well. We experiment with the attacker's strategy that treats all uncracked sweetwords as honeywords. The server uses the $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG model to generate honeywords, and the trawling attacker accordingly uses the strategy in Sec. IV to instantiate probabilistic models. Fig. 2 shows that when 20% uncracked sweetwords are selected randomly, $\mathcal{A}$'s success number is close to the ideal case (where all sweetwords are cracked); when 20% strongest sweetwords are marked as uncracked according to the Zxcvbn PSM [59], $\mathcal{A}$ can even gain a higher advantage than the ideal case. This is because Zxcvbn [59] helps eliminate these top 20% strongest sweetwords which are more likely to be produced by password models as honeywords. We obtain similar results no matter the uncracked sweetwords are selected locally (per account) or globally (in the whole sweetword file).

Summary. We, for the first time, propose a series of theoretic honeyword guessing models, each of which is based on varied kinds of capabilities allowed to an attacker. Particularly, we are the first to consider realistic attackers that know the order of user registration. These models enable us to design effective experiments with real-world datasets to evaluate the strength of a honeyword-generation method. *All this pushes the evaluation of honeywords toward statistical rigor, and also inspires the design of robust honeyword-generation methods.*

TABLE V
AN OVERVIEW OF THE DESIGN AND EVALUATION SPACE OF OUR NEW HONEYWORD-GENERATION METHODS.[†]

Attacker type (see Table I)	Our proposed method		How best to evaluate (i.e., compute the probabilities in Sec. III)	
	Password models used	Solutions to challenges	How best to instantiate $\Pr_{\text{PW}}(\cdot)$	How best to instantiate $\Pr_{\text{HW}}(\cdot)$
Type $\mathcal{A}_1$	List	+1 smooth	List; +1 smooth, $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$=1 smooth	Same with our method
Type $\mathcal{A}_2$	TarList	+1 smooth	TarList; +1 smooth, $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$=1 smooth	Same with our method
Type $\mathcal{A}_3$	$\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG	+1 smooth; Internal training; $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\widetilde{\text{HW}}}(\cdot)}$ >20 tweak tail	List; +1 smooth, $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\widetilde{\text{HW}}}(\cdot)}$=1 smooth	Same with our method
Type $\mathcal{A}_4$	$\frac{1}{3}$TarList+$\frac{1}{3}$TarMarkov +$\frac{1}{3}$TarPCFG	+1 smooth; Internal training; $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\widetilde{\text{HW}}}(\cdot)}$ >20 tweak tail	TarList; +1 smooth, $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\widetilde{\text{HW}}}(\cdot)}$=1 smooth	Same with our method

[†]The hybird method xList+yMarkov+zPCFG (x, y, z ∈[0,1] and x+y+z=1) means: x fraction of honeywords are from List model, y from PCFG, etc.

IV. OUR NEW CONSTRUCTIONS

We now elaborate on our new construction techniques of honeywords. Inspired by the above best "swords" (including the attacking theories and experiments), we forge the corresponding "shields"—four secure and efficient honeyword-generation methods based on various representative password models (e.g., trawling guessing models [40], [58] and targeted guessing models [56]). However, *the use of these password models is not straightforward, but requires significant, novel and creative efforts, and we show this through a series of exploratory investigations*. Besides, we manage to resolve several previously unexplored challenges that arise in the practical deployment of a honeyword method.

A. Overview of our new constructions

The real-password related design approach in [35] have two fundamental limitations. First, it easily reveals the features (e.g., length and character composition) of the real password PW_i to the attacker $\mathcal{A}$, once $\mathcal{A}$ has offline recovered a single sweetword from the password hash file F. What's worse, it is inherently *unable* to produce k-1 honeywords with the equal probability of $\Pr(\text{PW}_i)$, because that user passwords follow the Zipf's law (see Case 1 in Sec. III-A). Thus, we prefer the real-password *un*related design approach.

We consider four types of distinguishing attackers as listed in Table I, and design one best honeyword method for each of them. There is no single silver bullet. Our basic idea is that, *under a different kind of attacker, the best attack will be different; to resist this different best attack, we need a different best honeyword method*. This results in our four methods in Table V. The probabilistic password models (e.g., PCFG [40] and Targeted-PCFG [56]) cannot be readily applied, but require a number of tunings in both the design and evaluation process. These tunings are particularly challenging when $\mathcal{A}$ is with user registration order. The techniques in Table V and their underlying rationales will be elaborated in what follows.

B. Exploratory experiments

With the attacking theories (see Eqs. 4, 7∼9 in Sec. III-B), we now investigate *which models with what tunings (parameters)* can best withstand a given type of attackers.

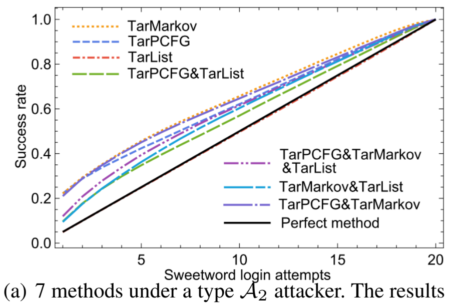
For attackers of type-$\mathcal{A}_1$. In this case, Eq. 4 applies. As there are three kinds of major password models (i.e., List, Markov and PCFG), a total of $7(=\binom{3}{1}+\binom{3}{2}+\binom{3}{3})$ honeyword methods will arise from combining these 3 password models. We attack these 7 honeyword methods by using three different password models. Fig. 6 in Appendix E shows that the List-based method always parallels with the perfect method in terms of both success-number and flatness: Whichever password model is used to instantiate $\Pr_{\text{PW}}(\cdot)$ in Eq. 4,[2] $\mathcal{A}$ can only distinguish about $526{=}10^4/19$ passwords (see Fig. 6(a)∼6(c)); $\mathcal{A}$ only gains a 5% success rate with one guess against 20 sweetwords (see Fig. 6(d)∼6(f)). Moreover, the List-based attacks are the most effective among three password models.

All this indicates that: (1) We shall prefer the List-model based honeyword method to instantiate $\Pr_{\text{HW}}(\cdot)$ when subject to a type-$\mathcal{A}_1$ attacker; and (2) Wang et al.'s proposal [53] of using the hybrid method $\frac{1}{3}$List+ $\frac{1}{3}$Markov+$\frac{1}{3}$PCFG to resist a $\mathcal{A}_1$ is *not* optimal. We note that, when $\mathcal{A}$ does not use List-model based attacks (see Figs. 6(e) and 6(f)), Markov or $\frac{1}{3}$List+$\frac{1}{3}$Markov+ $\frac{1}{3}$PCFG based methods sometimes perform better than the perfect method and the List method. This does *not* contradict with our preference but only emphasizes that $\mathcal{A}$ is ineffective when does not use List-model based attacks.

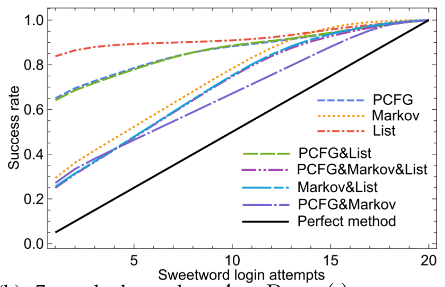
We note that when $\mathcal{A}$ uses the List-based password model to instantiate $\Pr_{\text{PW}}(\cdot)$, there will be a number of sweetwords that are with a large $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$, yet these sweetwords are not real passwords. We further investigate the issue and find that this is caused by the "+1" smoothing technique (proposed by [53]): If $\text{sw}_{i,j} \notin \mathcal{D}$, set $\Pr(\text{sw}_{i,j}){=}\frac{1}{|\mathcal{D}|+1}$. Wang et al. [53] have experimented with three smoothing methods (i.e., Laplace, Good-Turing and +1), and found the +1 method most effective. This kind of smoothing is suitable for attacking popular passwords, which is the case for Juels-Rivest's methods [35]. However, special attention shall be given to our methods where unpopular passwords are vulnerable (see Appendix D). For these extremely unpopular passwords, $\frac{1}{|\mathcal{D}|+1}$ is still too large and will result in a large $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$, causing a false positive. We devise a smoothing technique for $\mathcal{A}$ in such cases: If $\text{sw}_{i,j} \notin \mathcal{D}$ and $\frac{\Pr_{\text{PW}}(\text{sw}_{i,j})}{\Pr_{\text{HW}}(\text{sw}_{i,j})} > 1$, then set $\frac{\Pr_{\text{PW}}(\text{sw}_{i,j})}{\Pr_{\text{HW}}(\text{sw}_{i,j})}$=1. As a result, these false positives can be eliminated.

For attackers of type-$\mathcal{A}_2$. $\mathcal{A}$ now further exploits user PII as compared to a type-$\mathcal{A}_1$ attacker, and the Eq. 7 applies. Thus, the corresponding method shall be able to capture the PII semantics in passwords. This leads to our design of the TarList method to best resist against a type-$\mathcal{A}_2$ attacker. The rationale is that the TarList method inherits the merits of the

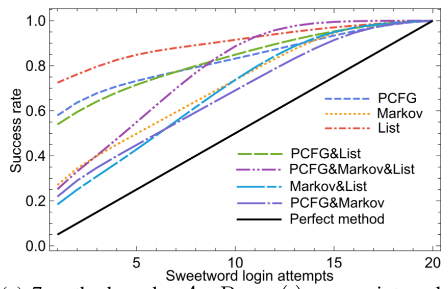
[2]To be most effective, $\mathcal{A}$ shall always use the system's honeyword method $\Pr_{\text{HW}}(\cdot)$, which is public info, to instantiate her $\Pr_{\text{HW}}(\cdot)$.



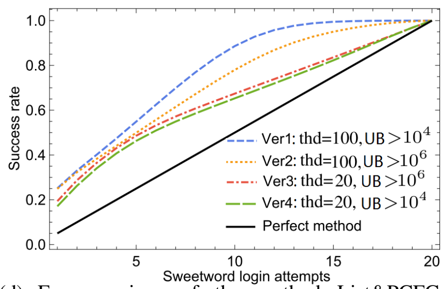
(a) 7 methods under a type $\mathcal{A}_2$ attacker. The results are desirable and the experiment succeeds.



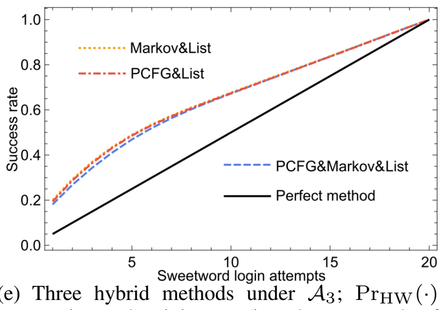
(b) 7 methods under $\mathcal{A}_3$; $\Pr_{\text{HW}}(\cdot)$ uses an external train set, without $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$>thd smooth.



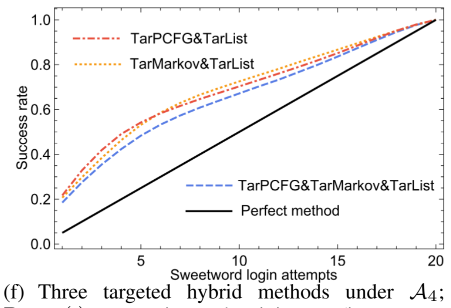
(c) 7 methods under $\mathcal{A}_3$; $\Pr_{\text{HW}}(\cdot)$ uses an internal training set, with no $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$ check.



(d) Four versions of the method List&PCFG &Markov (=$\frac{1}{3}$List+ $\frac{1}{3}$Markov+$\frac{1}{3}$PCFG) that tackle the $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$>thd smooth, UB=user base.



(e) Three hybrid methods under $\mathcal{A}_3$; $\Pr_{\text{HW}}(\cdot)$ uses an internal training set (i.e., the passwords of the target site) and $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$ >20 smooth.



(f) Three targeted hybrid methods under $\mathcal{A}_4$; $\Pr_{\text{HW}}(\cdot)$ uses an internal training set (i.e., passwords of the target site) and $\frac{\Pr_{\text{PW}}(\cdot)}{\Pr_{\text{HW}}(\cdot)}$>20 smooth.

Fig. 3. Exploratory experiments examining practical issues under attackers $\mathcal{A}_2 \sim \mathcal{A}_4$, trained on Dodonew-tr and tested on Dodonew-ts. TarList under $\mathcal{A}_2$, $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG under $\mathcal{A}_3$, and $\frac{1}{3}$TarList+$\frac{1}{3}$TarMarkov+$\frac{1}{3}$TarPCFG under $\mathcal{A}_4$ are the best ones.

List method as discussed above and can further deal with user PII. As expected, Fig. 3(a) shows that the optimal attacker only achieves a 5% success rate with one guess when k=20. $\mathcal{A}$ can only distinguish 531 real passwords when $\mathcal{T}_2$=10^4, quite close to that of the perfect method (i.e., 526=$10^4/19$). Hereafter we give $\mathcal{A}$'s success-number (at 10^4 failed attempts) instead of the whole graph due to space constraints.

For attackers of type-$\mathcal{A}_3$. $\mathcal{A}$ now further exploits the user registration order as compared to a type-$\mathcal{A}_1$ attacker, and Eq. 8 applies. With the knowledge of user registration order, $\mathcal{A}$ now can figure out which sweetwords are popular or not (at a given time point). As revealed in Appendix D, the List-based password model alone is vulnerable to unpopular passwords. For example, if $\mathcal{A}$ finds a sweetword that has never appeared in the earlier users' sweetlists, then she can be certain that this sweetword is the current user's real password. Thus, the List honeyword method is unsuitable for a type-$\mathcal{A}_3$ attacker. Fortunately, at the same time we find that the PCFG-based and Markov-based password models are good at capturing unpopular passwords, even though they each has their own defects (see Appendix D). This leads to our design of the hybrid method $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG to best resist a type-$\mathcal{A}_3$ attacker. For hybrid models, the way to instantiate the best strategy for the type-$\mathcal{A}_3$ attacker is similar to attacker $\mathcal{A}_1$; that is, the type-$\mathcal{A}_3$ attacker uses the smoothed List model to instantiate her *password* model, and uses the same honeyword generation model as the server to instantiate her *honeyword* model. In particular, $\mathcal{A}$ adaptively updates her honeyword model using the sweetword file to increase her advantage.

However, there are a number of practical issues to be addressed when applying the hybrid-model based honeyword design approach, and the two most challenging ones are: (1) Can we use external password datasets to be the training set when the user base is not large?; and (2) What can we do when encountering a sweetword $\text{sw}_{i,j}$ such that $\frac{\Pr_{\text{PW}}(\text{sw}_{i,j})}{\Pr_{\text{HW}}(\text{sw}_{i,j})} \gg 1$?

We now investigate the influence of external training datasets. For example, suppose a start-up web service wants to adopt a honeyword system when it only has 10^4 users. Generally, such a small user-base is considered insufficient to be used as training sets for password models like PCFG [40] and TarPCFG [56]. Thus, it is natural to employ an external training set. However, Fig. 3(b) shows that such an approach is insecure: Under $\mathcal{A}_3$, $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG only achieves 0.2525-flatness when the external Tianya training set is used. The reason is that: The external dataset is static, while the password distribution of the service under study is dynamic as new users register, and as time goes on, these two password distributions will be evidently different. $\mathcal{A}_3$ can exploit this fact. We prefer only using the internal training set, that is, the password dataset of its own users. As shown in Fig. 3(c), things go better (ϵ-flatness goes down by 5%$\sim$10% in general), but the situation is still undesirable.

Fortunately, as discussed in Sec. IV-A, these sweetwords x resulting in a large $\frac{\Pr_{\text{PW}}(x)}{\Pr_{\text{HW}}(x)}$ are mainly unpopular ones (i.e. with frequency f<10). This makes it reasonable to switch to the tweaking-tail method which is good at producing flat honeywords when x is unpopular: For password x, if we find $\frac{\Pr_{\text{PW}}(x)}{\Pr_{\text{HW}}(x)}$ is larger than a threshold thd, we use the tweaking-tail method. Note that, when distinguishing a sweetword x, $\mathcal{A}$ shall also switch to Eq. 3 when finding that $\frac{\Pr_{\text{PW}}(x)}{\Pr_{\text{HW}}(x)}$>thd. Now, how to set thd? After a number of experiments, we find that thd=20 performs the best. Fig. 3(d) presents an illustration.

When user U_i registers, the site first produces k-1 honeywords using our hybrid method $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG trained on the internal training set, and then U_i's password is inserted into the training set for update. Note that all password models we consider in this work can be trained in a streaming fashion, and thus there is no need to temporarily keep plaintext passwords on the server for training. Fig. 3(e) shows that, after addressing these practical issues, our hybrid method can be 0.178-flat (resp. 0.2525-flat in Fig. 3(b)) against $\mathcal{A}_3$ when k=20. $\mathcal{A}_3$ can only distinguish 736 real passwords when $\mathcal{T}_2$=10^4, suggesting our method is promising against $\mathcal{A}_3$.

For attackers of type-$\mathcal{A}_4$. $\mathcal{A}$ now further exploits user PII as compared to a type-$\mathcal{A}_3$ attacker, and the Eq. 9 applies. This leads to our design of the $\frac{1}{3}$TarList+$\frac{1}{3}$TarMarkov+$\frac{1}{3}$TarPCFG method to best resist $\mathcal{A}$. The rationale is the same with the case for a type-$\mathcal{A}_2$ attacker. As expected, the most harsh attacker only achieves a success rate of 18.2% with one guess against a list of 20 sweetwords (see Fig. 3(f)), and recovers mere 981 real passwords when the global threshold $\mathcal{T}_2$=10^4.

DoS attack. As our honeyword methods can produce honeywords that are nearly indistinguishable from real passwords, the fraction of popular honeywords would be close to the fraction of popular passwords. Therefore, when server S fails to adopt proper mitigations, a DoS attacker could trigger false alarms by deliberately submitting popular passwords. As discussed in Sec. II-B, S can effectively mitigate DoS threats by using blocklists, PSMs, rate-limiting and customized alarm policies, etc.

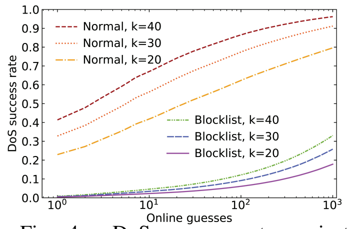


Fig. 4. DoS success rate against $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG. Trained on Dodonew-tr, tested on Dodonew-ts. "Blocklist" means a 100k blocklist is used to filter weak passwords (and honeywords), while "Normal" means no blocklist is used. "DoS success rate" means the probability of hitting $\mathcal{T}_1$=1 honeyword for each account.

We now conduct two preliminary experiments to show the effect of blocklists against DoS attacks. We first construct a blocklist with 10^5 popular passwords for Chinese users according to [56]; a blocklist of size 10^5 is widely recommended [19], [31]. To test the DoS mitigation effectiveness of our blocklist on the $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG method, we filter passwords that appear in the blocklist to simulate the deployment of blocklist on registration, and similarly block weak honeywords in the honeyword generation phase. Fig. 4 shows that introducing a blocklist significantly alleviates the DoS risks: When $\mathcal{T}_1$=1, with 100 online guesses the DoS attacker can achieve a success rate of 6.08% when k=20 and a success rate of 12.13% when k=40, while this figure will be 0.003% for k=20 and 0.025% for k=40 when $\mathcal{T}_1$=3 (see Fig. 4 in Appendix E). This indicates that a proper blocklist can effectively mitigate DoS attacks in a large part. Further coupled with PSMs, stricter rate-limiting and customized alarm policies, DoS threats can be further mitigated.

Model extraction attack. For the attacker that can somehow obtain the adaptive training model (e.g., compromise S), it is possible to extract high entropy passwords (e.g., fullname+birth year) directly from the model without online and offline guessing. Nevertheless, this risk is very limited. First, real user passwords are only used in training, and they can be deleted from the memory/disk once honeyword models are generated/updated. Second, $\mathcal{A}$ still has to generate a set of password guesses from (smoothed) password models, and invest considerable efforts to perform offline guessing; otherwise, it is impossible to know *which password* belongs to *which user account*. Finally, our honeyword system remains robust even when all sweetwords are recovered.

Summary. We retool probabilistic password cracking models to build flat honeywords. This approach has significant benefits in that: Future improvements to password models (e.g., deep learning) can be included easily into our honeyword methods. We manage to overcome several previously unexplored challenges that arise in the practical adoption of password models. This resolves the question of "can the password models underlying cracking algorithms (e.g., PCFG [58]) be easily adapted for use" as left in Juels-Rivest's work [35].

V. Evaluation Results

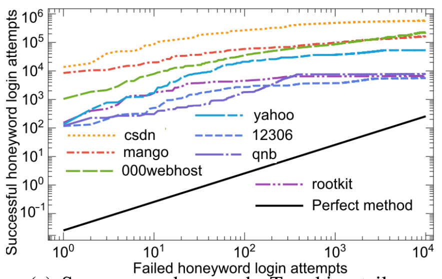
We now examine the scalability of our methods, and evaluate their security by both experiments and user-studies.

A. *Scalability with varying k*

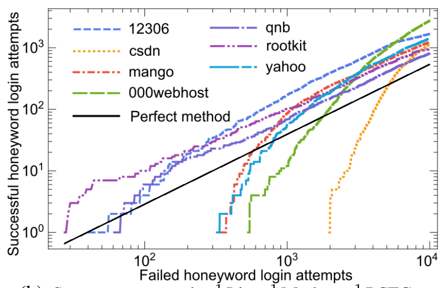
Clearly, the security of a honeyword method depends on the parameter k which indicates how many sweetwords are associated with each account. In Juels-Rivest's work [35], k is recommended to be 20, as they believed that it is acceptable for the attacker $\mathcal{A}$ to gain "a chance of at most 5% of picking the correct password" when given 20 sweetwords (i.e., being ϵ=0.05 flat). Existing literature only evaluates the situation of k=20. Now a natural question arises: How can we set k to ensure that the method achieves an expected security level (e.g., 0.05-flat)? In other words, how will a method perform with varying k? We call this property as a method's scalability.

As shown in Fig. 8(a) in Appendix E, a security goal of 0.05-flat seems prohibitively far away: Storing too many sweetwords for each user will not only increase storage cost but also delay login time. Though our hybrid method $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG (under $\mathcal{A}_3$) shows much better scalability (see Fig. 8(b)), it only reaches 0.1-flat when k=200. Note that ϵ decreases rather slowly as k increases. Interestingly, we find that ϵ and k well follow $\epsilon = a + \frac{b}{k^c}$, where a=0.084, b=1.4468, c=0.8329. As $k \to +\infty$, ϵ decreases monotonically and $\epsilon \to a$=0.084>0.05. This suggests that, for some password distributions, *0.05-flat is likely out of practical reach.*

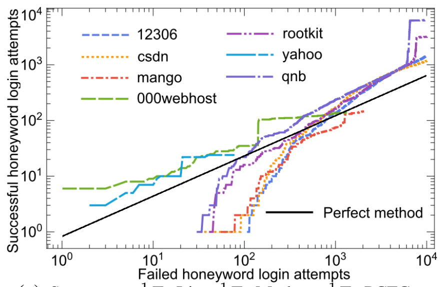
As shown in Fig. 8(b), our method $\frac{1}{3}$List+$\frac{1}{3}$Markov +$\frac{1}{3}$PCFG reaches ϵ=0.20 when k=20, ϵ=0.17 when k=30, ϵ=0.15 when k=40, and ϵ=0.14 when k=50. We have experimented with other combinations, and got similar diminish returns. Since services that adopt honeywords generally would be security-critical, we recommend k=40 to



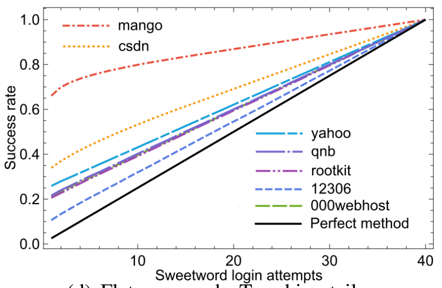
(a) Success-number graph: Tweaking-tail.



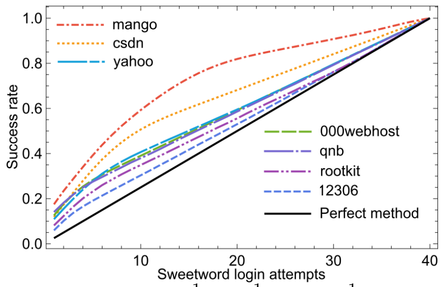
(b) Success-num graph: $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG.



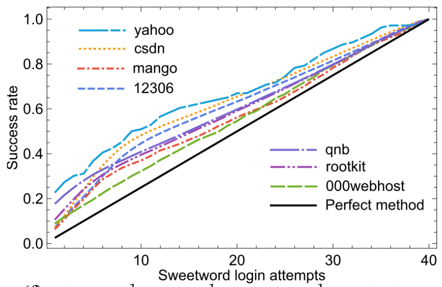
(c) Succ-num: $\frac{1}{3}$TarList+$\frac{1}{3}$TarMarkov+$\frac{1}{3}$TarPCFG.



(d) Flatness graph: Tweaking-tail.



(e) Flatness graph: $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG.



(f) Flatness: $\frac{1}{3}$TarList+$\frac{1}{3}$TarMarkov+$\frac{1}{3}$TarPCFG.

Fig. 5. Evaluating tweaking-tail [35] (under $\mathcal{A}_1$) and our two methods (under $\mathcal{A}_3$ and $\mathcal{A}_4$, respectively) by using seven password datasets. 50% of each dataset is used as training and another 50% as testing. Our methods achieve 0.15-flat under harsh attackers.

ensure an acceptable level of security (i.e., 0.15-flat) while being reasonably cost-effective as shown in Sec. V-B.

B. Empirical evaluation results

Note that our List method and TarList method will always yield the same security results as a perfect method under type-$\mathcal{A}_1$ and $\mathcal{A}_2$ attackers, respectively. This is because the optimal attacker needs to employ the List password model $\Pr_{\mathcal{D}}(\cdot)$ to compute Eq. 3, and the server employs the $\Pr_{\mathcal{D}'}(\cdot)$ to generate honeywords, while $\mathcal{A}$'s training set D can only approximate but will never equal the target site's password distribution D'. This has been empirically shown in Figs. 6 and 3. Thus, we mainly evaluate our remaining two methods.

To avoid overfitting, the datasets used in all above exploratory experiments will not be used here. For fair evaluation, we perform attacks simulating a type-$\mathcal{A}_1$ attacker against tweaking-tail, attacks simulating a type-$\mathcal{A}_3$ attacker against $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG, and attacks simulating a type-$\mathcal{A}_4$ attacker against $\frac{1}{3}$TarList+$\frac{1}{3}$TarMarkov+$\frac{1}{3}$TarPCFG. This is because each method is designed under a specified security model, and it is only sensible to claim some form of security under that model. Fig. 5 shows that for most of the datasets, our methods can be 0.15-flat, while this figure for tweaking-tail is 0.2^{+} and actually, our methods are evaluated under much harsher attackers. Particularly, in our methods there are orders of magnitude less "low-hanging fruits"(see Figs. 5(b)$\sim$5(c)) that can be obtained by $\mathcal{A}$. In all, *our empirical results well accord with the exploratory experiments, and when setting k=40, our methods can ensure the security level of 0.15-flat.*

Discussion. Fig. 5 shows that the language, service type and size of the training/testing datasets all have a significant influence on honeyword security. This is in line with [53], [55]. There is no sign from our evaluation results that passwords/honeywords from English users are more vulnerable or secure than those of Chinese users. Moreover, the perceived risk (service type) of the site will greatly influence users' password behaviors. This has been confirmed in password research [32], [55]. For instance, users on the QNB banking site (see Table II) have, on average, less-common passwords: The top 10 most popular passwords only account for 0.59% of QNB users, while the figure for the remaining 10 datasets (in Table II) is 4.16% on average. Thus, the closer the training set is to the passwords of the target site, the better/flatter the generated honeywords will be. This explains why adaptive password/honeyword models are preferable. Results of QNB in Fig. 5 demonstrate that our adaptive honeyword methods are suitable for highly sensitive services like Banking.

Overheads. The overall overheads of our methods are low and acceptable, because: (1) All training, generation and update processes are conducted on the server side and need *no* user interactions/feedbacks, and thus they do not impair user experience; (2) Training is costly, but it is only conducted once; and (3) Generation and update processes mainly entail some table lookups, which is lightweight. For instance, when k=40 and trained on 32M Rockyou, training costs 70 min on a common PC; the generation time is 2ms; honeyword storage for 10^7 users costs 12.8GB when using PBKDF2-SHA256.

C. Human-based evaluation results

While our methods can provide desirable security against computer-automated attackers, whether the conclusion still holds under semantic-aware humans is unknown. This is of particular concern when considering that, there are many semantics in passwords (e.g., `bond007` and `john1981`) that can be easily recognized by a human being but difficult to be understood by an automated attacker. Thus, we recruited 11 graduate students who were taking a "network security" seminar to participate in our evaluation.

Setups. Before starting the experiments, our participants were asked to read the honeyword-related literature [21], [24], [35], and were informed of both Juels-Rivest's four [35] and our four honeyword-generation methods. One month later, they were asked to take a quiz with questions covering various aspects of honeywords, and all passed. Their expertise enables them to comprehend all the twelve attacking scenarios inside out, and to know clues/weaknesses and where and how to look for them in telling honeywords and real passwords apart. Hence, they are competent adversaries in our setting. For the sake of incentive, we specify that, when given a list of k sweetwords, 1 CNY will be awarded if a participant finds the real password with the 1st attempt, $\frac{1}{2}$CNY with the 2nd attempt, and so on. The procedure of human experiments is established with the help of two usable-security researchers with survey expertise.

The experiment lasted six consecutive days during a holiday. On each day, one attacking scenario is accomplished in the morning and another in the afternoon. During each scenario, a participant will be given 40 sweetword lists, each of which includes 20 sweetwords, and participants are asked to finish within 30 minutes. This leads to a total consumption of 5280(=40×2×6×11) user accounts (with PII), which are randomly drawn from the 161,517 PII-associated Dodonew password accounts (see Table IV). The reason why we choose Dodonew as the main dataset in this experiment is given in Appendix E. We had initially attempted to include 40 sweetwords in each sweetword list (and one scenario costs about 45 minutes), yet feedbacks from the two usable-security experts signal that this is too fatiguing. The schedule was established and sent to every participants before the experiment.

In the testing phase, they came to our lab to conduct the attacks. Each computer stores a dozen of password datasets that the participant can query, which simulates a basic attacker with the type-$\mathcal{A}_1$ capability. For each attacking scenario, either the method or attacker type will be different from the other ones. For a type-$\mathcal{A}_1$ attacker, participants are only given 40 sweetword lists. For $\mathcal{A}_2$, the common PII of each victim will be provided; for $\mathcal{A}_3$, the order of the sweetword list is just the order of user registration; for $\mathcal{A}_4$, this is the joint case of $\mathcal{A}_2$ and $\mathcal{A}_3$. For ethical considerations, all computers are disconnected from the Internet, no paper or memory device are allowed for recording, and Email suffix and NID are not given to the participants to make the users less identifiable.

Results. Each sub-figure in Fig. 9 in Appendix E shows the flatness curves for all 11 experts (denoted by A to K) under a given attacking scenario. As summarized in Table VI, the four methods in [35] achieve 0.40^+-flatness under Type-$\mathcal{A}_1$ attacker and 0.48^+-flatness under Type-$\mathcal{A}_2$ attacker, far from perfect flatness. In comparison, both List and $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG methods achieve almost perfect flatness (i.e., $\epsilon \approx \frac{1}{20}$) under non PII-aware attackers. Even when attackers are PII-aware (i.e., Type-$\mathcal{A}_2$ and $\mathcal{A}_4$), our corresponding methods still achieve 0.09^--flatness. This suggests that our targeted methods can well capture user PII semantics. As compared to the four methods in [35], all our four methods are over 4.5=(0.4023/0.0886) times more secure in terms of ϵ-flatness. To sum up, results suggest that our methods are substantially better at resisting human-expert attackers.

Generally, when human experts are not provided with the victim user's PII, they are considerably more effective than computer-automated algorithms (see Table X of [53]). For instance, when the victims' PII is not available, human experts achieve a success rate of 40.23%∼55.00% (with just one guess) at telling apart real passwords from honeywords generated by Juels-Rivest's four methods [35], while the figure for computer-automated algorithms is 34.21%∼49.02%. When human experts are provided with victims' PII, their advantages are comparable to PII-enriched computer-automated algorithms. For instance, when the victims' PII is available, human experts achieve a success rate of 58.64%∼71.59% (with just one guess) at telling apart real passwords from honeywords generated by Juels-Rivest's four methods [35], while the figure for computer-automated algorithms is 56.80%∼67.90%.

TABLE VI
ϵ-FLAT INFORMATION UNDER HUMAN ATTACKS.

Honeyword-generation methods	Attacker type	ϵ-flatness
Tweak tail	Type-$\mathcal{A}_1$	0.4023
Tweak tail	Type-$\mathcal{A}_2$	0.5864
Model-syntax	Type-$\mathcal{A}_1$	0.5500
Model-syntax	Type-$\mathcal{A}_2$	0.7159
Hybrid	Type-$\mathcal{A}_1$	0.4886
Hybrid	Type-$\mathcal{A}_2$	0.6023
Simple model	Type-$\mathcal{A}_1$	0.4682
Simple model	Type-$\mathcal{A}_2$	0.6659
List	Type-$\mathcal{A}_1$	**0.0568**
TarList	Type-$\mathcal{A}_2$	**0.0705**
$\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG	Type-$\mathcal{A}_3$	**0.0591**
$\frac{1}{3}$TarList+$\frac{1}{3}$TarMarkov+$\frac{1}{3}$TarPCFG	Type-$\mathcal{A}_4$	**0.0886**

The human-expert attacks on Dodonew have well shown that Chinese human attackers are PII-aware in nature, and since English users and Chinese users show quite similar PII usage behaviors [55], it is highly likely that attackers in other languages would have similar performance. Thus, when user PII is available, honeywords shall be generated with PII.

Summary. Our empirical evaluation builds on 11 large-scale datasets and considers various attackers. Further, to see how our methods perform under sematic-aware humans, we conduct a user study of 11 trained expert attackers. Results show that they can survive both automated and human attacks.

VI. Conclusion

We have systematically tackled the question of how best to attack, design and evaluate honeyword-generation methods. For the first time, we provided theoretical proofs and empirical explorations of how best to attack honeywords. This in-depth understanding of honeyword attackers enables us to suggest a suite of honeyword-generation methods by using leading probabilistic password models. We demonstrated the effectiveness of our methods by conducting both automated experiments and trained human-expert attacks. In the meanwhile, we addressed two open problems left in [35] and one in [53].

Acknowledgment

The authors are grateful to the shepherd and anonymous reviewers for their invaluable comments. Ding Wang is the corresponding author, and part of the work was done while he was at Peking University. This research was in part supported by the National Natural Science Foundation of China under Grant No.62172240. There is no competing interests.

References

[1] *All Data Breach Sources*, Oct. 2018, https://breachalarm.com/all-sources.

[2] *Yahoo Raises Breach Estimate to Full 3 Billion Accounts, By Far Biggest Known*, Oct. 2017, http://fortune.com/2017/10/03/yahoo-breach-mail/.

[3] *The Password is Dead, Long Live the Password!*, October 2016, https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2016/october/the-password-is-dead-long-live-the-password/.

[4] F. Adowsett, *What has been leaked: impacts of the big data breaches*, April 2016, https://rantfoundry.wordpress.com/2016/04/19/what-has-been-leaked-impacts-of-the-big-data-breaches/.

[5] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee, "Pasta: password-based threshold authentication," in *Proc. CCS 2018*, pp. 2042–2059.

[6] A. Akshima, D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya, "Generation of secure and reliable honeywords, preventing false detection," *IEEE Trans. Depend. Secur. Comput.*, vol. 16, no. 5, pp. 757–769, 2019.

[7] M. H. Almeshekah, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford, "Ersatzpasswords: Ending password cracking and detecting password leakage," in *Proc. ACSAC 2015*, pp. 311–320.

[8] M. Andrey, *Building a Distributed Network in the Cloud: Using Amazon EC2 to Break Passwords*, Aug. 2017, https://blog.elcomsoft.com/2017/08/breaking-passwords-in-the-cloud-using-amazon-p2-instances/.

[9] A. J. Aviv, D. Budzitowski, and R. Kuber, "Is bigger better? comparing user-generated passwords on 3x3 vs. 4x4 grid sizes for android's pattern unlock," in *Proc. ACSAC 2015*, pp. 301–310.

[10] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: new generation of memory-hard functions for password hashing and other applications," in *Proc. EuroS&P 2016*, pp. 292–302.

[11] J. Blocki, B. Harsha, S. Kang, S. Lee, L. Xing, and S. Zhou, "Data-independent memory hard functions: New attacks and stronger constructions," in *Proc. CRYPTO 2019*, pp. 573–607.

[12] J. Blocki, B. Harsha, and S. Zhou, "On the economics of offline password cracking," in *Proc. IEEE S&P 2018*, pp. 35–53.

[13] L. Bloomberg, *50 Million MyFitnessPal Accounts Have Been Hacked, Under Armour Says*, March 2018, http://fortune.com/2018/03/29/myfitnesspal-password-under-armour-data-breach/.

[14] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proc. IEEE S&P 2012*, pp. 538–552.

[15] J. Bonneau, C. Herley, P. Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE S&P 2012*, pp. 553–567.

[16] J. Bonneau, C. Herley, P. van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Commun. ACM*, vol. 58, no. 7, pp. 78–87, 2015.

[17] M. Burnett, *Is there life after passwords?*, July 2016, https://medium.com/un-hackable/is-there-life-after-passwords-290d50fc6f7d.

[18] J. Camenisch, A. Lehmann, and G. Neven, "Optimal distributed password verification," in *Proc. ACM CCS 2015*, pp. 182–194.

[19] U. N. C. S. Centre, *How password deny lists can help your users to make sensible password choices*, April 2019, https://www.ncsc.gov.uk/blog-post/passwords-passwords-everywhere.

[20] A. Chaabane, G. Acs, and M. Kaafar, "You are what you like! information leakage through users' interests," in *Proc. NDSS 2012*.

[21] N. Chakraborty and S. Mondal, "Few notes towards making honeyword system more secure and usable," in *Proc. ACM SIN 2015*, pp. 237–245.

[22] R. Chatterjee, A. Athalye, D. Akhawe, A. Juels, and T. Ristenpart, "Password typos and how to correct them securely," in *Proc. IEEE S&P 2016*, pp. 799–818.

[23] C. Cimpanu, *26 million LiveJournal credentials leaked online, sold on the dark web*, May 2020, https://www.zdnet.com/article/26-million-livejournal-credentials-leaked-online-sold-on-the-dark-web/.

[24] I. Erguler, "Achieving flatness: Selecting the honeywords from existing user passwords," *IEEE Trans. Depend. Secur. Comput.*, vol. 13, no. 2, pp. 284–295, 2016.

[25] S. Furnell and R. Esmael, "Evaluating the effect of guidance and feedback upon password compliance," *Comput. Fraud Secur.*, vol. 2017, no. 1, pp. 5–10, 2017.

[26] X. Gao, Y. Yang, C. Liu, C. Mitropoulos, J. Lindqvist, and A. Oulasvirta, "Forgetting of passwords: Ecological theory and data," in *Proc. USENIX SEC 2018*, pp. 221–238.

[27] J. Goldman, *Chinese Hackers Publish 20 Million Hotel Reservations*, Dec. 2013, http://www.esecurityplanet.com/hackers/chinese-hackers-publish-20-million-hotel-reservations.html.

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

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

[30] Y. Guo, Z. Zhang, and Y. Guo, "Superword: A honeyword system for achieving higher security goals," *Comput. Secur.*, vol. 103, p. 101689, 2021.

[31] H. Habib, J. Colnago, W. Melicher, B. Ur, S. Segreti, L. Bauer, N. Christin, and L. Cranor, "Password creation in the presence of blacklists," *Proc. USEC 2017*, pp. 1–11.

[32] A. Hanamsagar, S. S. Woo, C. Kanich, and J. Mirkovic, "Leveraging semantic transformation to investigate password habits and their causes," in *Proc. ACM CHI 2018*, pp. 1–10.

[33] *Resetting passwords to keep your files safe*, Aug. 2016, https://blogs.dropbox.com/dropbox/2016/08/resetting-passwords-to-keep-your-files-safe/.

[34] A. Holmes, *533 million Facebook users' phone numbers and personal data have been leaked online*, Apr. 2021, https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4.

[35] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proc. ACM CCS 2013*, pp. 145–160.

[36] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun, "Sound-proof: usable two-factor authentication based on ambient sound," in *Proc. USENIX SEC 2015*, pp. 483–498.

[37] K. Krawiecka, A. Kurnikov, A. Paverd, M. Mannan, and N. Asokan, "Safekeeper: Protecting web passwords using trusted execution environments," in *Proc. WWW 2018*, pp. 349–358.

[38] *Researcher Finds Credentials for 92 Million Users of DNA Testing Firm MyHeritage*, June 2018, https://krebsonsecurity.com/2018/06/researcher-finds-credentials-for-92-million-users-of-dna-testing-firm-myheritage/.

[39] R. W. Lai, C. Egger, M. Reinert, S. S. Chow, M. Maffei, and D. Schröder, "Simple password-hardened encryption services," in *Proc. USENIX SEC 2018*, pp. 1405–1421.

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

[41] S. Mamonov and R. Benbunan-Fich, "The impact of information security threat awareness on privacy-protective behaviors," *Comput. Hum Behav.*, vol. 83, pp. 32–44, 2018.

[42] M. L. Mazurek, S. Komanduri, T. Vidas, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur, "Measuring password guessability for an entire university," in *Proc. ACM CCS 2013*, pp. 173–186.

[43] W. Melicher, B. Ur, S. M. Segreti, 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*.

[44] C. Morris, *Massive data leak exposes 700 million LinkedIn users information*, June 2021, https://fortune.com/2021/06/30/linkedin-data-theft-700-million-users-personal-information-cybersecurity/.

[45] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why do developers get password storage wrong?: A qualitative usability study," in *Proc. ACM CCS 2017*, pp. 311–328.

[46] P. Negi, P. Sharma, V. Jain, and B. Bahmani, "K-means++ vs. Behavioral biometrics: One loop to rule them all," in *Proc. NDSS 2018*, pp. 1–15.

[47] T. Pham, *Four Years Later, Anthem Breached Again: Hackers Stole Credentials*, Feb. 2015, http://duo.sc/2ene0Pr.

[48] B. Ur, "Supporting password-security decisions with data," Ph.D. dissertation, Carnegie Mellon University, 2016.

[49] *Passwords are not lame and they're not dead*, Aug. 2017, https://it.toolbox.com/blogs/itmanagement/passwords-are-not-lame-and-theyre-not-dead-heres-why-072417.

[50] H. Varun, *Qatar National Bank Suffers Massive Breach*, April 2016, http://www.bankinfosecurity.com/qatar-national-bank-suffers-massive-breach-a-9068.

[51] D. Votipka, R. Stevens, E. M. Redmiles, J. Hu, and M. L. Mazurek, "Hackers vs. testers: A comparison of software vulnerability discovery processes," in *Proc. IEEE S&P 2018*, pp. 134–151.

[52] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inform. Foren. Secur.*, vol. 12, no. 11, pp. 2776–2791, 2017.

[53] D. Wang, H. Cheng, P. Wang, J. Yan, and X. Huang, "A security analysis of honeywords," in *Proc. NDSS 2018*, pp. 1–15.

[54] D. Wang, D. He, H. Cheng, and P. Wang, "fuzzyPSM: A new password strength meter using fuzzy probabilistic context-free grammars," in *Proc. IEEE/IFIP DSN 2016*, pp. 595–606, http://bit.ly/2ahJ8CO.

[55] 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.

[56] 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.

[57] K. C. Wang and M. K. Reiter, "How to end password reuse on the web," in *Proc. NDSS 2019*, pp. 1–15.

[58] 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.

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

Appendix

A. Detailed information about our datasets

We evaluate the existing honeyword methods and our proposed ones based on 11 large real password datasets (see Table III), a total of 105.44 million passwords. Three of our datasets were leaked in MD5 hash, and we manage to recover an overwhelming fraction of them by using off-the-shelf cracking tools like Hashcat and John the Ripper, various trawling guessing models [40] and the targeted guessing model TarGuess [56], on a common PC with GPU in one week. More specially, Rootkit initially consists of 71,228 passwords and we recover 97.46% of them; QNB initially contains 97,415 passwords and was leaked from the Qatar national bank, which is located in Middle East, in April 2016 [50], and we recover 79,580 (81.69%) of them; Mango initially contains 1,113,638 passwords, and we recover 96.51% of them.

Particularly, four password datasets (i.e., 12306, ClixSense, Rootkit and QNB) are associated with various kinds of PII as shown in Table III. To facilitate a more comprehensive empirical analysis of honeyword security under targeted attackers, we further match the *non*-PII-associated password datasets with these PII-associated password datasets by using email. As a result, this produces *nine* PII-associated password datasets as shown in Table IV: (1) The four Chinese PII-associated datasets are obtained by matching the corresponding *non*-PII-associated dataset with 12306; (2) The four US-English ones are: PII-Rootkit, PII-ClixSense, and two other ones obtained by matching 000webhost and Yahoo with ClixSense, respectively; and (3) PII-QNB, which is QNB itself. Note that, the *non*-PII-associated US-English dataset Rockyou includes neither email nor NID, and thus it cannot be matched.

We further employ two auxiliary PII datasets that do not have passwords (i.e., Hotel and 51job) to augment each Chinese password dataset to obtain more PII-associated accounts by matching email or NID. We note that many PII-associated accounts miss some important PII attributes, and they can be supplemented by using the auxiliary PII datasets.

All our 11 datasets were hacked and made public on the Internet between 2009 and 2016, and they may be a bit old. However, they can represent current user password behaviors due to three reasons. First, three datasets (i.e., 000webhost, ClixSense, and Qatar national bank (QNB)) were leaked after 2015 and may well exhibit up-to-date user password behaviors. Second, human-being's cognition capabilities (e.g., memory) remain rather stable as time goes on, and Bonneau has revealed that "passwords have changed only marginally since then (1990)" [14]. Finally, the password ecosystem evolves very slowly. A number of recent researches (see [16], [25], [28], [55]) show that password guidance and practices implemented on leading sites have seldomly changed over time.

B. Revisiting recent honeyword methods

Here we give a brief analysis of the pitfalls in three recent honeyword proposals [6], [24], [30]. Due to space constraints, readers are referred to the companion site of this revision https://github.com/honeyword/honeywords-project for more details. Briefly, the computational cost of Honeyindex is $k/2$ times larger (on average) than Honeyword [35]. In addition, Honeyindex [24] suffers from the mapping attack and the peeling-onions style distinguishing attack; the latter can be resisted by regenerating sweetindexes periodically, yet this will bring a high probability of false alarms. For the "evolving password model" by Akshima et al. [6], it is inherently vulnerable to the "normalized top-PW" attack mentioned in Sec. III-A; their "user-profile model" is further vulnerable to targeted attackers. As for Superword [30], it is prone to DoS attack, which cannot be remedied with these DoS measures mentioned in Sec. II-B. Besides, Superword [30] has a large communication overhead, and introduces new vulnerabilities by placing too much burden on the honeychecker which defeats the purpose of Honeywords in the first place.

C. Proofs of Theorem 1, 2, and 3

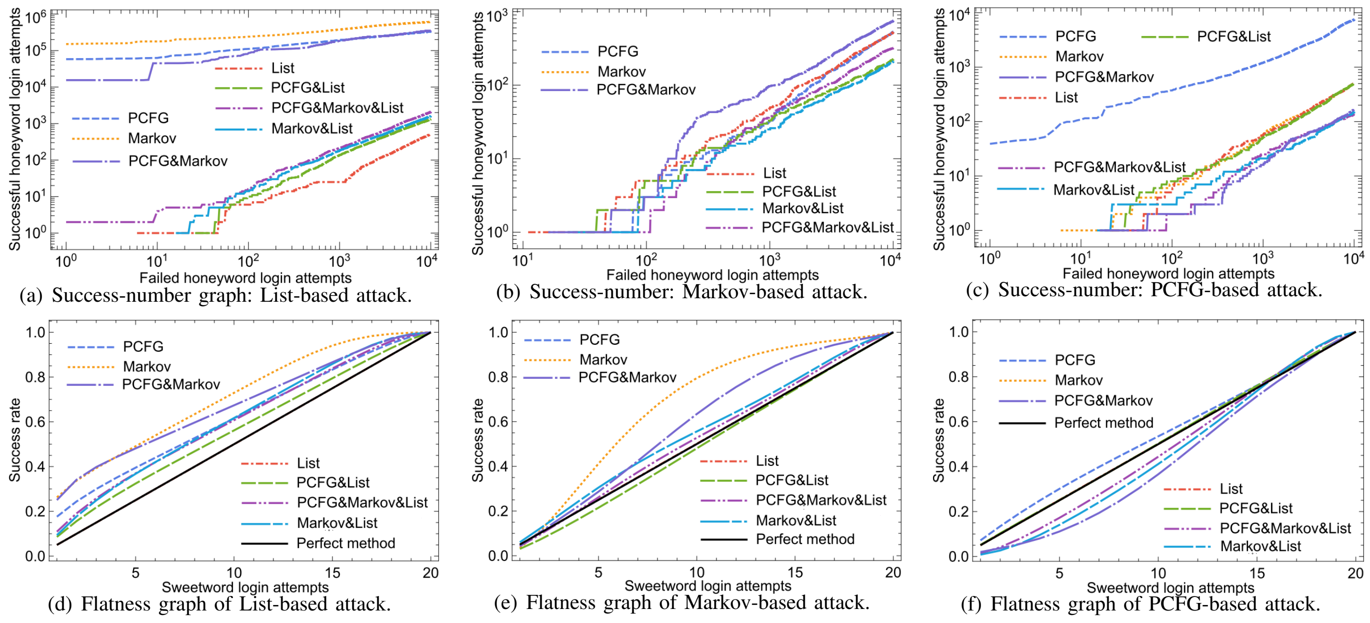
Theorem 1: Let $\mathsf{hw}_{i,j}$ ($1 \leq j \leq k$) denote the event that $sw_{i,j}$ is produced as a sweetword for U_i, and $\mathsf{pw}_{i,j}$ denote the event that U_i selects $sw_{i,j}$ as her real password. We have

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i) = \frac{\Pr(\mathsf{pw}_{i,j})\prod_{l\neq j}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j})}{\sum_{t=1}^{k}\Pr(\mathsf{pw}_{i,t})\prod_{l\neq t}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t})},$$

under the assumption that $\mathsf{hw}_{i,1},\ldots,\mathsf{hw}_{i,j-1},\mathsf{hw}_{i,j+1},\ldots,\mathsf{hw}_{i,k}$ are *mutually* independent under the event $\mathsf{pw}_{i,j}$.

Proof. Since $\mathsf{hw}_{i,1},\ldots,\mathsf{hw}_{i,j-1},\mathsf{hw}_{i,j+1},\ldots,\mathsf{hw}_{i,k}$ are mutually independent under the event $\mathsf{pw}_{i,j}$, we have $\Pr(\mathrm{SW}_i|\mathsf{pw}_{i,j}) = \prod_{l\neq j}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j})$. Now, leveraging the Bayesian theory, we can derive:

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i) = \frac{\Pr(\mathrm{SW}_i|\mathsf{pw}_{i,j})\cdot\Pr(\mathsf{pw}_{i,j})}{\sum_{t=1}^{k}\Pr(\mathrm{SW}_i|\mathsf{pw}_{i,t})\cdot\Pr(\mathsf{pw}_{i,t})}$$
$$=\frac{\Pr(\mathsf{pw}_{i,j})\cdot\prod_{l\neq j}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j})}{\sum_{t=1}^{k}\Pr(\mathsf{pw}_{i,t})\cdot\prod_{l\neq t}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t})}. \qquad \blacksquare$$



(a) Success-number graph: List-based attack. (b) Success-number: Markov-based attack. (c) Success-number: PCFG-based attack.
(d) Flatness graph of List-based attack. (e) Flatness graph of Markov-based attack. (f) Flatness graph of PCFG-based attack.

Fig. 6. Experiment results for attacking 7 honeyword methods with 3 different password guessing models under attacker-$\mathcal{A}_1$, trained on Dodonew-tr and tested on Dodonew-ts. The List method performs *equally well* with the perfect method: Their flatness-graph lines overlap with each other. When $\mathcal{A}_1$ does not use List-model based attacks, Markov or hybrid methods sometimes perform better than other methods.

Theorem 2: Let F denote the event that the file F is produced as the password-file for all users, and the other definitions comply with those in Theorem 1. We have

$$\Pr(\mathsf{pw}_{i,j}|F) = \Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i),$$

under the assumptions that users independently create passwords, and the assumptions of Theorem 1.

Proof. Since users create their own passwords independently and SW_i only depends on U_i's real password $\mathsf{pw}_{i,j}$, $\mathrm{SW}_1, \ldots, \mathrm{SW}_n$ will be mutually independent, we can derive:

$$\begin{aligned}&\Pr(\mathsf{pw}_{i,j}|F)\\ =&\Pr(\mathsf{pw}_{i,j}, \begin{pmatrix} sw_{1,1} & \cdots & sw_{1,k} \\ \vdots & \ddots & \vdots \\ sw_{n,1} & \cdots & sw_{n,k}\end{pmatrix})/\Pr(\begin{pmatrix} sw_{1,1} & \cdots & sw_{1,k} \\ \vdots & \ddots & \vdots \\ sw_{n,1} & \cdots & sw_{n,k}\end{pmatrix})\\ =&\frac{\Pr(\mathsf{pw}_{i,j}, \mathrm{SW}_i)\prod_{l\neq i}\Pr(\mathrm{SW}_i)}{\prod_{l=1}^{n}\Pr(\mathrm{SW}_l)}\\ =&\frac{\Pr(\mathsf{pw}_{i,j}, \mathrm{SW}_i)}{\Pr(\mathrm{SW}_i)} = \Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i).\end{aligned}$$

■

Theorem 3: Let $\mathsf{pw}_{i,j}$ $(1 \leq j \leq k)$ denote the event that U_i selects $sw_{i,j}$ as her real password, $\mathsf{hw}_{i,j}$ denote the event that $sw_{i,j}$ is produced as a honeyword for U_i, and let PII denote U_i's PII. We have

$$\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{PII}) = \frac{\Pr(\mathsf{pw}_{i,j}|\mathsf{PII})\prod_{l\neq j}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j},\mathsf{PII})}{\sum_{t=1}^{k}\Pr(\mathsf{pw}_{i,t}|\mathsf{PII})\prod_{l\neq t}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t},\mathsf{PII})},$$

under the assumption that $\mathsf{hw}_{i,1}, \ldots, \mathsf{hw}_{i,j-1}, \mathsf{hw}_{i,j+1}, \ldots, \mathsf{hw}_{i,k}$ are *mutually* independent under the event $(\mathsf{pw}_{i,j}, \mathsf{PII})$.

Proof. Here we provide a detailed proof for Theorem 3. Since $\mathsf{hw}_{i,1}, \ldots, \mathsf{hw}_{i,j-1}, \mathsf{hw}_{i,j+1}, \ldots, \mathsf{hw}_{i,k}$ are mutually independent under the events $\mathsf{pw}_{i,j}$ and $(\mathsf{pw}_{i,j}, \mathsf{PII})$, we have $\Pr(\mathrm{SW}_i|\mathsf{pw}_{i,j}, \mathsf{PII}) = \prod_{l\neq j}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j}, \mathsf{PII})$. Now, leveraging the Bayesian theory, we can derive:

$$\begin{aligned}&\Pr(\mathsf{pw}_{i,j}|\mathrm{SW}_i, \mathsf{PII})\\ =&\frac{\Pr(\mathrm{SW}_i|\mathsf{pw}_{i,j}, \mathsf{PII})\cdot\Pr(\mathsf{pw}_{i,j}|\mathsf{PII})}{\sum_{t=1}^{k}\Pr(\mathrm{SW}_i|\mathsf{pw}_{i,t}|\mathsf{PII})\cdot\Pr(\mathsf{pw}_{i,t}|\mathsf{PII})}\\ =&\frac{\Pr(\mathsf{pw}_{i,j}|\mathsf{PII})\cdot\prod_{l\neq j}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,j}, \mathsf{PII})}{\sum_{t=1}^{k}\Pr(\mathsf{pw}_{i,t}|\mathsf{PII})\cdot\prod_{l\neq t}\Pr(\mathsf{hw}_{i,l}|\mathsf{pw}_{i,t}, \mathsf{PII})}.\end{aligned}$$

■

D. Pros and Cons of existing PW models

As mentioned in Sec. II-D, we mainly consider six representative, probabilistic password models: PCFG [58], Markov [40], List [53], and their targeted versions. The first question we are confronted with is: As there are a number of candidates, which password model shall be preferred? To answer it, we investigate the weaknesses of each individual model. Generally, the effectiveness of a machine-learning-based password model relies on two factors: The model itself and the training sets used. To preclude the impacts of training sets, as recommended in [53], [56], we randomly split the Dodonew dataset into two equal parts, and use part-1 (i.e., Dodonew-tr) for training and part-2 (i.e., Dodonew-ts) for testing. We implement the PCFG model and Markov model according to the most recent improvements in [40]. More specifically, for PCFG model the probabilities associated with letter segments are learned directly from the training process, and for Markov model we use a fourth order Markov chain with end-symbol normalization and Laplace smoothing.

Guided by Eq. 4, we know the defects of password model are the passwords pw, $\frac{\Pr_{\mathrm{PW}}(pw)}{\Pr_{\mathrm{HW}}(pw)} \gg 1$. These passwords will be cracked first by the optimal adversary. In Table VII, we measure the value of $\frac{\Pr_{\mathrm{PW}}(pw)}{\Pr_{\mathrm{HW}}(pw)}$ for typical passwords pw, where $\Pr_{\mathrm{PW}}(pw)$ comes directly from Dodonew-ts and $\Pr_{\mathrm{HW}}(pw)$ is output by each password model. We can conjecture that:

TABLE VII
THE VALUE OF $\frac{\Pr_{\mathrm{PW}}(\cdot)}{\Pr_{\mathrm{HW}}(\cdot)}$ GIVEN BY EACH PASSWORD MODEL FOR TYPICAL PWS (DODONEW-TR VS. DODONEW-TS).†

Typical password	Probability $\Pr_{\mathrm{PW}}(\cdot)$	List	PCFG	Markov
`123456`	0.01443750	0.99	1.25	0.96
`password`	0.00044136	1.02	1.63	1.25
`123qwe`	0.00027111	0.95	46.35	1.39
`1q2w3e4r`	0.00011588	1.14	$6.57 \cdot 10^{10}$	1.18
`147852369`	0.00004293	1.07	0.92	107.42
`110120130`	0.00002337	0.84	0.86	5059.23
`110011`	0.00000886	0.77	1.05	41.06
`password123`	0.00000381	1.07	0.55	1.82
`p@ssw0rd`	0.00000221	0.78	$8.74 \cdot 10^{10}$	1.25
`XX123456`	0.00000160	13.00	1.39	4.58
`34567890`	0.00000148	12.53	6.87	6.92
`123qwe123qwe`	0.00000123	0.77	6662.66	27.13
`Password123`	0.00000037	0.60	2.02	5.31
`iloveyou123456`	0.00000025	0.67	0.59	0.51
`123456abcdefg`	0.00000012	0.09	7.98	0.31
`520yong`	0.00000011	0.09	0.51	0.44

† A value of $\frac{\Pr_{\mathrm{PW}}(\cdot)}{\Pr_{\mathrm{HW}}(\cdot)}$ with dark gray means it's the *worst* one to approximate the real probability $\Pr_{\mathrm{PW}}(\cdot)$, and light gray means the *2nd worst* one.

The List model is good at approximating popular passwords, PCFG good at passwords with a simple structure, and Markov good at short passwords. All this suggest that each individual password model has it own advantages, and a hybrid model would be desirable when $\mathcal{A}$ (e.g., a type $\mathcal{A}_3$ attacker) may exploit each model's disadvantages.

Note that, for a hybrid model A&B that is resulted from models A and B, denoted by $\Pr_{\mathrm{AB}}(pw) = \frac{1}{2}\Pr_{\mathrm{A}}(pw) + \frac{1}{2}\Pr_{\mathrm{B}}(pw)$), the event $\frac{\Pr_{\mathrm{PW}}(pw)}{\Pr_{\mathrm{AB}}(pw)} \gg 1$ happens if and only if $\frac{\Pr_{\mathrm{PW}}(pw)}{\Pr_{\mathrm{A}}(pw)} \gg 1$ and $\frac{\Pr_{\mathrm{PW}}(pw)}{\Pr_{\mathrm{B}}(pw)} \gg 1$. So such hybrid models can significantly alleviate defects of individual password model. Therefore, we use hybird models (e.g., $\frac{1}{3}$List + $\frac{1}{3}$Markov + $\frac{1}{3}$PCFG) to resist type $\mathcal{A}_3$ and $\mathcal{A}_4$ attackers.

The above conjectures are corroborated by Table VIII. We measure the passwords that appear in the top-10^3 $\frac{\Pr_{\mathrm{PW}}(\cdot)}{\Pr_{\mathrm{HW}}(\cdot)}$ list under each model. According to our theories in Sec. III, these passwords will be attacked in $\mathcal{A}$'s first 1000 attempts and thus they are the top-1000 most vulnerable ones. As expected, all methods are not good at dealing with PII-semantic involved passwords and passwords not covered by the training set. *This outlines the need for designing PII-aware methods when type $\mathcal{A}_2$ attacker (i.e., with PII) is considered.*

TABLE VIII
BASIC INFO ABOUT PWS IN THE TOP-10^3 $\frac{\Pr_{\mathrm{PW}}(\cdot)}{\Pr_{\mathrm{HW}}(\cdot)}$ LIST UNDER EACH PW MODEL.

	List	Markov	PCFG
In top-10^4 PW list	**0.000**	**0.000**	**0.000**
Not in training set	**1.000**	**0.993**	**0.998**
Feq.>2 (in test set)	**1.000**	**0.006**	**0.005**
Feq.<10 (in test set)	0.960	1.000	1.000
Password len. ≥16	0.000	**0.906**	**1.000**
Structure len. ≥6	0.001	**0.995**	**0.576**
With semantic info	0.012	0.016	0.051
Email/site address	0.000	**0.442**	**0.695**

* feq.=frequencey; len.=length.

Table VIII also reveals some unexpected results. No matter honeywords are generated by which model, all these top-10^3 most vulnerable passwords are *not* popular ones—they do not fall into the top-10^4 popular password list. When combining the 3rd and 4th rows, one can infer that the List model is not good at predicting these passwords with a frequency $2<f<10$, while the other two models are not good at these with $f \leq 2$. This has important implications for designing these hybrid models: For a user password pw, if we find $\frac{\Pr_{\mathrm{PW}}(pw)}{\Pr_{\mathrm{HW}}(pw)}$ is dangerously high (i.e., ≫1), *we can switch to the tweaking-tail method which is good at producing flat honeywords when pw is unpopular.*

E. Additional experiments and discussions

Fig. 6 demonstrates the effectiveness of our List-model based honeyword method against three different password guessing models, under the basic attacker $\mathcal{A}_1$ who only has public datasets. $\mathcal{A}_1$ can merely distinguish about 526=$10^4/19$ passwords (see Fig. 6(a)∼6(c)); $\mathcal{A}$ gains a success rate of 5% with one guess against 20 sweetwords (see Fig. 6(d)∼6(f)). Interestingly, when $\mathcal{A}_1$ does not employ List-model based attacks, Markov or hybrid methods sometimes perform the best over other methods including the List-model based one. This indicates the importance of designing optimal attacks for a given scenario, otherwise the security might be overestimated.

Fig. 4 of Sec. IV-B shows that DoS can be largely mitigated by imposing a 10^5 blocklist that filters weak passwords and honeywords, under the alarm policy $\mathcal{T}_1$=1 that a single honeyword attempt against an account raises an alarm. Still, when allowed 100 online login attempts, the DoS attacker can achieve a success rate of 6.08% when k=20 and a success rate of 12.13% when k=40. The effectiveness is not very desirable. Fig. 7 further investigates the effectiveness of this same blocklist under the case when we set $\mathcal{T}_1$=3. Results show that this countermeasure significantly alleviates DoS risks: Within 100 guesses, the DoS attacker can only achieve a success rate of 0.003% when k=20, 0.010% when k=30, and 0.025% when k=40. In contrast, without this blocklist, this figure will be 5.45% when k=20, 14.64% when k=30, and 26.41% when k=40.

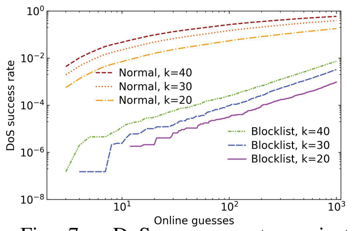
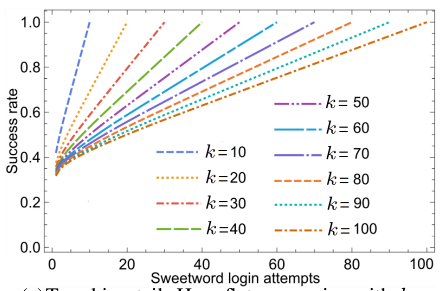


Fig. 7. DoS success rate against $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG. Trained on Dodonew-tr, tested on Dodonew-ts. "Blocklist" means a 100k blocklist is used to filter weak passwords (and honeywords), while "Normal" means no blocklist is used. "DoS success rate" means the probability of hitting $\mathcal{T}_1$=3 honeywords for each account.

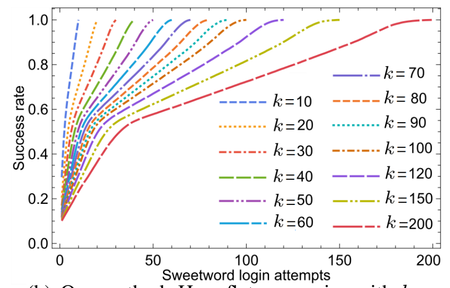
Fig. 8 illustrates how flatness varies with the number (i.e., k) of sweetwords that are associated with each user account. There are obvious diminish-returns: When k is large enough (e.g., ≥60), marginal security gains will be achieved when k is further increased. On the other hand, a larger k means a larger storage cost. Thus, we recommend k=40 to be cost-effective.

We choose Dodonew as the dataset used in our human-based experiments, because: (1) Dodonew is a canonical dataset for Chinese users, and it has been used in almost every research regarding passwords of Chinese users (see [12], [40], [52], [53], [56]); and (2) For ethics considerations—Dodonew was leaked in 2011, ten years ago, and it is reasonable to assume that Dodonew users have already changed their passwords.

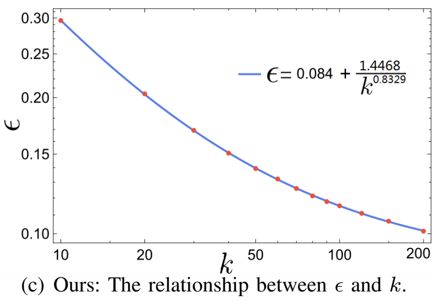
Fig. 9 shows the flatness curves of human-based evaluation, and detailed setups can be found in Sec. V-C. The four methods in [35] achieve 0.40^{+}-flatness under $\mathcal{A}_1$ and 0.48^{+}-flatness under $\mathcal{A}_2$, far from perfect. In comparison, both List and $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG methods achieve almost perfect flatness (i.e., $\epsilon\approx\frac{1}{20}$) under non PII-aware human attackers. Even when attackers are PII-aware (see Figs. 9(j) and 9(l)), our methods still achieve 0.09^{-}-flatness. This implies that our targeted methods can well capture user PII semantics.



(a)Tweaking-tail: How flatness varies with k

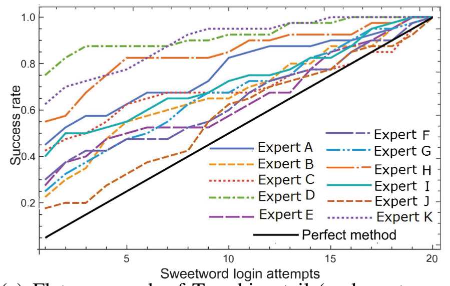


(b) Our method: How flatness varies with k.

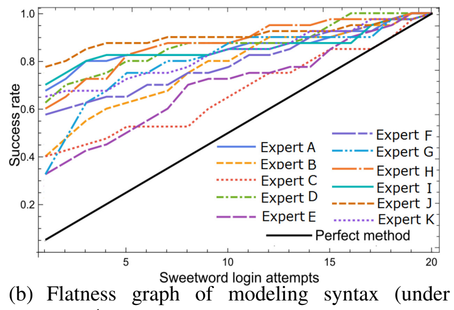


(c) Ours: The relationship between ϵ and k.

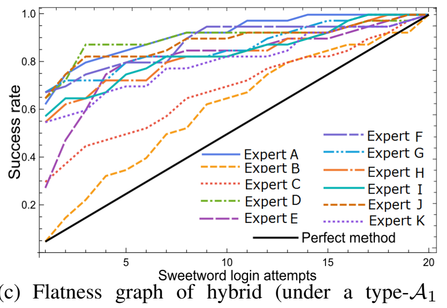
Fig. 8. How the flatness curve varies with k, trained on Dodonew-tr and tested on Dodonew-ts. Here we use tweaking-tail (under $\mathcal{A}_1$) and $\frac{1}{3}$List+$\frac{1}{3}$Markov+$\frac{1}{3}$PCFG (under $\mathcal{A}_3$) as examples. The sub-fig(c) shows how ϵ (=$y|_{x=1}$ in sub-fig(b)) in our hybrid method evolves with k.



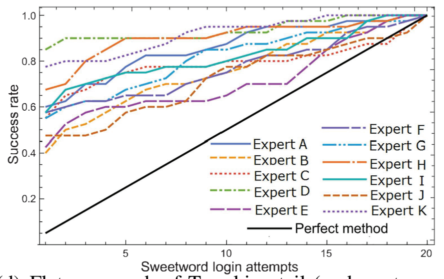
(a) Flatness graph of Tweaking-tail (under a type-$\mathcal{A}_1$ attacker).



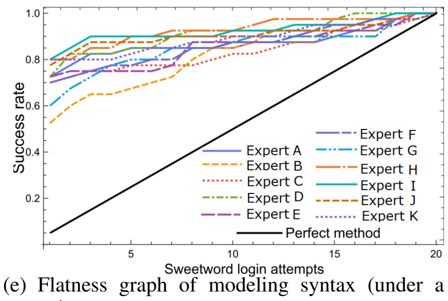
(b) Flatness graph of modeling syntax (under a type-$\mathcal{A}_1$ attacker).



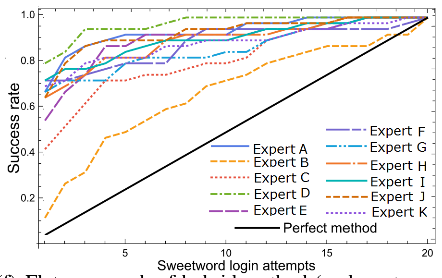
(c) Flatness graph of hybrid (under a type-$\mathcal{A}_1$ attacker).



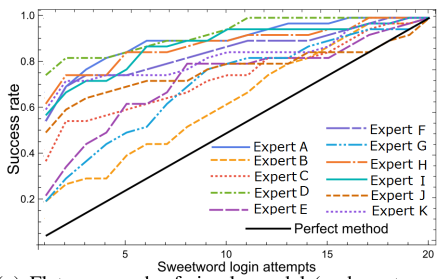
(d) Flatness graph of Tweaking-tail (under a type-$\mathcal{A}_2$ attacker).



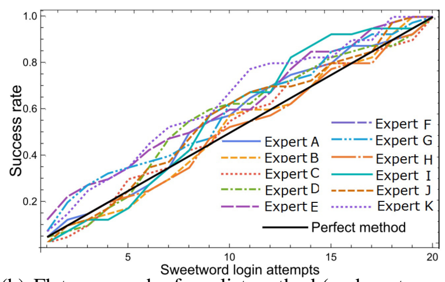
(e) Flatness graph of modeling syntax (under a type-$\mathcal{A}_2$ attacker).



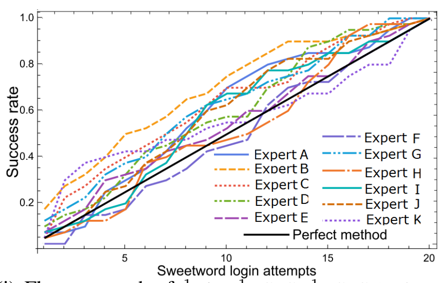
(f) Flatness graph of hybrid method (under a type-$\mathcal{A}_2$ attacker)



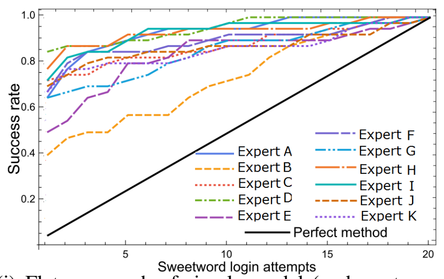
(g) Flatness graph of simple model (under a type-$\mathcal{A}_1$ attacker).



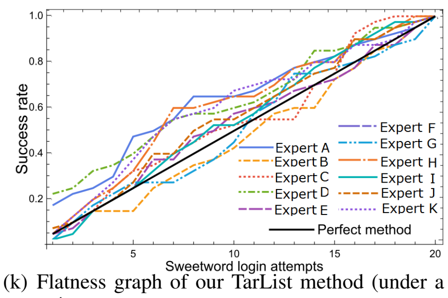
(h) Flatness graph of our list method (under a type-$\mathcal{A}_1$ attacker).



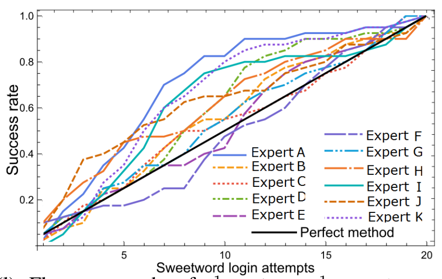
(i) Flatness graph of $\frac{1}{3}$List+$\frac{1}{3}$PCFG+$\frac{1}{3}$PCFG (under a type-$\mathcal{A}_3$ attacker).



(j) Flatness graph of simple model (under a type-$\mathcal{A}_2$ attacker).



(k) Flatness graph of our TarList method (under a type-$\mathcal{A}_2$ attacker).



(l) Flatness graph of $\frac{1}{3}$TarList + $\frac{1}{3}$TarMarkov + $\frac{1}{3}$TarPCFG (under a type-$\mathcal{A}_4$ attacker).

Fig. 9. Evaluating our methods and Juels-Rivest's ones [35] by using *human-expert-based* attacks. "under $\mathcal{A}_x$" means experts are simulating type-$\mathcal{A}_x$ attackers. Humans are particularly effective at telling apart real PWs generated by Juels-Rivest's methods [35] when given PII (see sub-figs d, e, f and j), yet they show no advantages over computer-based attackers against our methods. The 5,280 tested accounts are from PII-Dodonew. All our four methods show significantly better security than Juels-Rivest's four real-password related methods [35].