A Security Analysis of Honeywords

Ding Wang, Haibo Cheng, Ping Wang
Peking University
{wangdingg, chenghaibo, pwang}@pku.edu.cn

Jeff Yan
Linköping University
jeff.yan@liu.se

Xinyi Huang
Fujian Normal University
xyhuang81@gmail.com

***Abstract*—Honeywords are decoy passwords associated with each user account, and they contribute a promising approach to detecting password leakage. This approach was first proposed by Juels and Rivest at CCS'13, and has been covered by hundreds of medias and also adopted in various research domains. The idea of honeywords looks deceptively simple, but it is a deep and sophisticated challenge to automatically generate honeywords that are hard to differentiate from real passwords. In Juels-Rivest's work, four main honeyword-generation methods are suggested but only justified by *heuristic* security arguments.**

In this work, we for the first time develop a series of practical experiments using 10 large-scale datasets, a total of 104 million real-world passwords, to *quantitatively* evaluate the security that these four methods can provide. Our results reveal that they all fail to provide the expected security: real passwords can be distinguished with a success rate of 29.29%∼32.62% by our basic trawling-guessing attacker, but not the expected 5%, with just *one* guess (when each user account is associated with 19 honeywords as recommended). This figure reaches 34.21%∼49.02% under the advanced trawling-guessing attackers who make use of various state-of-the-art probabilistic password models. We further evaluate the security of Juels-Rivest's methods under a targeted-guessing attacker who can exploit the victim' personal information, and the results are even more alarming: 56.81%∼67.98%. Overall, our work resolves three open problems in honeyword research, as defined by Juels and Rivest.

I. Introduction

Passwords firmly remain the most prevalent method for user authentication and are likely to keep their place in the foreseeable future, despite their notorious defects in both security and usability [3], [6], [30]. An inherent limitation of the existing password-based authentication systems is that the server need maintain a sensitive file comprised of passwords of all registered users, and this file provides attackers/insiders with a rich target for compromise. These days it is no news to hear that high-profile web services have been compromised and millions of passwords were leaked, and some quite recent victims include Yahoo [18], Dropbox [20], Last.fm, LinkedIn, Weebly [27], and MySpace [31], to just name a few.

What's most disturbing is that, these breaches were often detected only when the attackers had well exploited the data and then posted (or sold) it online, which is generally *months, even years* after the breach initially occurred. For instance, the most recent catastrophic password breach revealed on October 2017 involves the entire 3 billion Yahoo userbase, yet the breach actually had occurred *four years* before [18]; The Weebly password breach revealed in Oct. 2016 involves 43 million users, and users were asked to change passwords, yet the breach actually had occurred *eight months* before [27]; The 68 million Dropbox breach occurred in 2012, yet users are asked to change passwords *four years* later when the stolen data surfaced in the public in May 2016 [20]; The detection of the 360 million Myspace dataset took *eight years*: this dataset was compromised in 2008 [31] yet only detected until May 2016, when its sale information was posted online. The 2016 data breach report by Verizon also reveals that, among the 2,260 breaches investigated, over 85% were first detected by external parties, 91% took weeks to detect, 70% took months even years to detect [13]. *All this highlights the imperative need for active, timely password-breach detection methods to enable responsive counter-actions*.

Even if the leaked passwords are stored in salted-hash as standard practice, this poses no real obstacle for an attacker to recover them by an overwhelming percentage by using modern machine-learning based cracking algorithms [24], [30] and common hardware like GPUs (see [12]). Sophisticated hash functions (e.g., bcrypt) are designed to slow down the attacker's cracking process by the same factor as the honest server is willing to spend on password verification, but the attacker is likely to be better equipped with dedicated password-cracking hardware [15]. Thus, once the password hash file is obtained by an attacker, it is realistic to assume that most of them can be offline guessed. In addition, since the attacker only need perform guessing locally, slow hash functions do not facilitate the detection of password-file leakage.

Recently, there have been approaches proposed to completely eliminate the possibility of offline password guessing: (1) using a machine-dependent function (e.g., ErsatzPasswords [2]); (2) employing distributed cryptography (e.g., threshold password-authenticated secret sharing [7]); and (3) using external password-hardening services (e.g., Phoenix [23]). However, all these approaches require substantial changes to the server-side authentication systems. Besides, the first approach does not support backing up password hash files in a distributed manner, and thus it is unsuitable for Internet-scale sites due to its poor scalability; The second necessitates client-side system changes, which is not user-friendly and widely deemed not desirable; The third is subject to a single point of failure and may leak user behavior information to external parties.

A more promising approach to improving the situation, first proposed by Juels and Rivest in 2013 [21], is to introduce decoy passwords (called "honeywords") to associate with each user's account (see Fig. 1). Their intriguing idea is that, even if an attacker $\mathcal{A}$ has stolen the password file and recovered all the passwords, she has to first tell apart the user's real password from a set of $k-1$ (e.g., k=20 suggested in [21]) intentionally generated honeywords. These $k-1$ honeywords and the real password are unifiedly called k "sweetwords". When honeywords are well generated, to figure out the real

Network and Distributed Systems Security (NDSS) Symposium 2018
18-21 February 2018, San Diego, CA, USA
ISBN 1-891562-49-5
http://dx.doi.org/10.14722/ndss.2018.23142
www.ndss-symposium.org

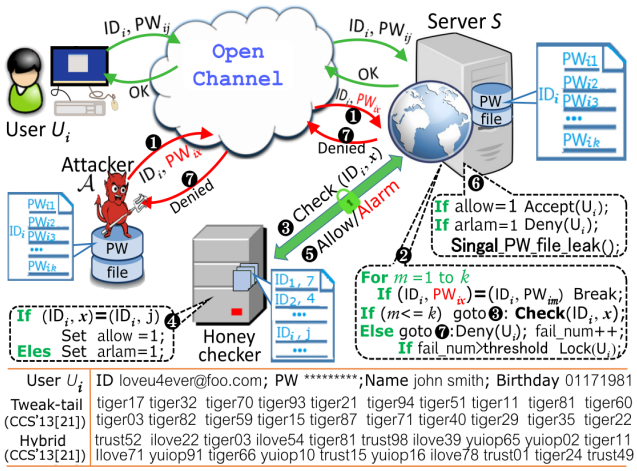


Fig. 1. Password (PW) authentication with honeywords. For better illustration, here passwords are shown in plain-text, while in reality they are stored in salted hash. The bottom of the figure shows some personal information about the victim user U_i, and exemplifies two sets of 19(=k-1) honeywords generated for U_i's password "tiger81" by two different methods in [21]: tweaking-tail and hybrid (see Sec. III-A).

password, $\mathcal{A}$ has to perform a few online login attempts by using the server as a verification oracle. Such online login attempts would not only significantly impede attackers [11], but also set off an alarm of password-file compromise at the server when a honeyword is attempted for login. This approach involves relatively few changes to the existing server-side system and no changes to the client-side system, and thus it seems rather practical. It has attracted hundreds of medias and also been adopted in various research domains (e.g., graphical passwords [33] and cryptographic protocol [29]).

A. Challenges and motivations

A main challenge, as discussed in [7], [14], [21], is how to generate honeywords that cannot be easily distinguished from real passwords. In Juels-Rivest's work [21], five honeyword-generation methods are suggested: four for the legacy user-interface (UI) and one for the modified-UI, and these legacy-UI methods are preferred due to usability advantages. Thus, we also focus on honeyword generation for the legacy-UI.

All the Juels-Rivest methods are random-replacement based, and thus inherently unable to resist semantic-aware attackers as shown by *some* typical counter-example passwords (e.g., `bound007` and `john1981`) in the literature [10], [14]. Yet, to what extent these methods are effective (or ineffective) against semantic aware/un-aware attackers has never been theoretically or empirically quantified.

Most prior art on honeywords (see [10], [14], [21]) *merely provide heuristic security arguments when evaluating a method, neither a rigorous theoretical analysis nor an empirical evaluation with real-world datasets was given*. As we will show with realistic experiments, all the honeyword-generation methods in [21] largely fail to provide the expected security, and real passwords can be distinguished with high success rates. An improvement named "honeyindex" was given in [14], yet its evaluation is still heuristic-based and it suffers from the "peeling-onions style" distinguishing attack and critical deployment issues.[1] Golla et al. [17] empirically showed how the Kullback-Leibler divergence (KLD) metric can be used to distinguish a real password distribution from a decoy one, yet this metric is unsuitable for the honeyword-attacking scenario where we need to distinguish the single, real password from a set of *distinct* sweetwords. In all, existing honeyword methods (see [14], [21]) are short of a sound evaluation, and whether they can achieve the claimed security is unknown.

What's more, *little attention has been given to the analysis of honeyword security under targeted guessing attackers*. A targeted attacker (see [30]) exploits not only users' behavior of selecting popular passwords (e.g., `abc123`, `iloveyou`) but also the victims' personally identifiable information (PII) like name and birthday. To our knowledge, existing works [10], [14], [21] mainly consider trawling guessing attackers (see [5], [24], [32]) who exploit only users' behavior of choosing popular passwords. However, a large fraction of users build passwords using their own PII. For instance, as we will show in Sec. II-A, 51.43% of Dodonew normal users, 27.16% of Qutar national bank users and 12.76% of Rootkit hackers employ their own, common PII to build passwords.

This user behavior is increasingly vulnerable, as users' PII can now be easily learned from social networks and unending data breaches [16], [18], [26]. For example, the recent large-scale data breach in Oct. 2017 involves 3 billion Yahoo users, with their names, phone numbers, birthdates leaked [18]; In July 2017, the largest credit-reporting agency in America, Equifax, leaked PII data about half of the US population, including names, birthdates, SSN [25]; The April 2016 PII breach against Turkishens involves 64% of the total population [1]. *As the targeted guessing threat is becoming increasingly realistic, honeywords shall be evaluated under this new threat.*

B. Our contributions

In this work, we make the following key contributions:

- **Trawling guessing attacks**. We for the first time develop a series of experiments using large-scale real password data to evaluate the four honeyword-generation methods in [21], and find that they all fail to provide the expected security. We show that real passwords can be distinguished with a success rate of 29.29%∼32.62%, but not the expected 5%, with just *one* guess under a basic, trawling guessing attacker (when each user account is associated with 19 honeywords as recommended, i.e., the parameter $k = 20$ [21]). This figure reaches 34.21%∼49.02% under the advanced trawling-guessing attackers who make use of various state-of-the-art probabilistic password cracking models (e.g., Markov [24] and PCFG [32]).
- **Targeted guessing attacks**. To see how Juels-Rivest's methods perform under semantic-aware attackers, we for the first time evaluate the security of honeywords by performing targeted guessing attacks. We show that real passwords can be distinguished with a success rate of 56.8%∼67.9% by performing just *one* guess (when $k = 20$), if the attacker knows some common personal information like name and birthday of the victim user. This answers the question of "how well can targeted attacks help identify users' passwords for particular honeyword-generation methods" as left in [21].
- **Extensive evaluation.** Our experiments build on various leading probabilistic password cracking models

[1] The companion site: https://github.com/pkusec/rethinking-honeywords

(e.g., Markov [24], TarGuess [30] and PCFG [32]). To make our results as generic as possible, we employ 10 large-scale real-world password lists, which consist of a total of 104.36 million passwords, covering various popular web services. This is the first study that empirically evaluates honeywords. Our extensive evaluation suggests that Juels-Rivest's methods can survive neither PII-unaware nor PII-aware attackers.

- **New insights.** We obtain a number of insights, some expected and some surprising, from our empirical experiments. We reveal that generating decoy passwords (by randomly replacing parts/whole of the real password) to be equally probable with the user's real password is *inherently impossible*. This indicates that Juels-Rivest's random-replacement based approach is inherently vulnerable, which is unexpected. We also show that probabilistic password models *cannot* be readily employed to generate honeywords, which is opposed to common belief (as hold in [21]). As expected (and confirmed by us), these password models can be used as building blocks to design effective experiments with real-world datasets to evaluate a honeyword method. This answers the open question of "are there good experimental methods for quantifying the flatness of honeyword methods" in [21].

II. Datasets, Security Model and Metrics

A. Our datasets

We evaluate Juels-Rivest's honeyword methods based on 10 large real password datasets (see Table I), including four from Chinese sites and six from English sites. In total, our datasets are composed of 104.36 million plain-text passwords and involve 9 different web services. Besides some early disclosed datasets (e.g., Rockyou and Dodonew) which have been widely used in research [8], [24], [30], we also incorporate three very recently leaked datasets that may exhibit up-to-date user password behaviors. These datasets were compromised by hackers or leaked by insiders, and were publicly available on the Internet for some time. Since the canonical dataset Rockyou only contains passwords (with no user names or emails), and it will not be used for evaluating targeted threats. The role of each dataset will be specified where necessary.

TABLE I. 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	
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.

Two of our datasets were leaked in MD5 hash, and we manage to recover an overwhelming fraction of them by using various trawling guessing models [24] as well as the targeted guessing model TarGuess [30] on a common PC with GPU in one week. More specially, Rootkit initially consists of 71,228 passwords and we manage to recover 97.46% of them; QNB initially contains 97,415 passwords [19], and we manage to recover 79,580 (81.69%) of them. The QNB dataset was leaked from the Qatar national bank, which is located in Middle East, in April 2016 [19]. To our knowledge, this is the first real-world banking-password dataset that is explored in an academic study.

TABLE II. Basic info 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.

Particularly, four password datasets (i.e., 12306, ClixSense, Rootkit and QNB) are associated with various kinds of PII as shown in Table II. To facilitate a more comprehensive empirical analysis of honeyword security under targeted attackers, we further match the *non*-PII-associated datasets with these PII-associated datasets by using email. As a result, this produces *nine* PII-associated password datasets as shown in the first row in of Table III: (1) the four Chinese ones 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 (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.

Table III demonstrates that users love to employ their personal information to build passwords. Here we measure the PII usages by using the *type-based* PII-tagging approach proposed in [30], for it has been shown much more accurate than other approaches. As high as 36.95%∼51.43% of Chinese users employ at least one of their six kinds of PII to construct passwords, while this figure for US-English users is 12.76%∼29.94% and for ME-English users is 27.16%. In comparison, the PII-associated US-English users show a more secure behavior in PII usages. This is expected, because they are all Rootkit hackers or ClixSense online cash-earning users (as resulted from email matches). In other words, *our PII-associated US-English users well represent technique-savvy users*. Our results show that *a large number of users build passwords using PII, and thus sound honeyword-generation methods shall take this user behavior into account*.

We highlight that though QNB users speak English, their passwords have little correlation with the other four English userbases. This is expected: QNB users are mainly from Middle East. Thus, we divide users into three groups: Chinese, US-English and ME-English. In all, *our corpus is new, comprehensive and well represents real-world password distributions, and to our knowledge, it is also among the largest and most diversified ones ever collected for use in a password study*.

TABLE III. PERCENTAGES OF USERS BUILDING PASSWORDS WITH THEIR *own* HETEROGENEOUS PERSONALLY IDENTIFIABLE INFORMATION (PII).†

Typical usages of personally identifiable information	PII-Tianya (430,966)	PII-Dodonew (161,517)	PII-12306 (129,303)	PII-CSDN (77,216)	PII-Rootkit (69,330)	PII-000web-host(153,390)	PII-ClixSense (2,222,045)	PII-Yahoo (16,307)	PII-QNB (77,799)
Name (7 subtypes, e.g., `johnsmith`, `john`, `jsmith`)	9.13%	23.82%	23.83%	16.71%	4.32%	12.66%	9.14%	6.95%	8.52%
Birthday (10 subtypes, e.g., `01171981`, `1981`, `0117`)	20.80%	16.37%	18.75%	19.16%	1.57%	7.36%	7.07%	5.20%	11.85%
Email_prefix (3 subtypes, e.g., `moon123`, `moon`, `123`)	6.31%	8.60%	6.61%	8.65%	3.75%	7.89%	5.14%	3.86%	4.91%
User name (3 types, e.g., `loveu1314`, `loveu`, `1314`)	3.09%	10.53%	10.12%	7.46%	3.45%	5.74%	5.29%	3.82%	7.61%
Phone # (3 subtypes, e.g., `4153022671`, `415`, `2671`)	1.18%	1.00%	0.89%	1.43%	–	–	–	–	1.92%
NID (3 subtypes, e.g., `620915337`, `620`, `5337`)	0.81%	0.39%	0.71%	0.12%	–	–	–	–	0.13%
Total personal information usages (all above)	36.95%	51.43%	50.71%	46.87%	12.76%	29.94%	24.81%	18.76%	27.16%

†The specific sub-types of each kind of PII we consider are the same with that of TarGuess-I [30]. For instance, 23.82% in the top left corner means that 23.82% of the 161,517 PII-associated Dodonew users employ at least one of their 7 sub-types of name information to build passwords.

B. Security model

In this paper, we mainly focus on the security that can be provided to the underlying user authentication system when honeywords are in place. Without loss of generality, we consider the most general case, i.e. the client-server architecture. We discuss the primary kinds of attacks which exploit honeywords and can be possibly launched against the honeyword system.

The honeyword system. As shown in Fig. 1, there are four entities involved: a user U_i, an authentication server S, a honeychecker, and an attacker $\mathcal{A}$. User U_i has registered an account (ID_i, PW_i) at S, and some PII may also be needed (e.g., Gmail registration requires name, birthday, phone and gender). On the server side, what's different from the traditional password authentication is that, S conducts a command $\mathsf{Gen}(k; \mathrm{PW}_i)$: S generates a list of k-1 distinct, plausibly looking decoy passwords (called *honeywords*) to associate with U_i's account, where $k = 20$ as recommended in [21]. Password PW_i and its k-1 honeywords are unifiedly called k *sweetwords*. Generally, there are two broad kinds of honeyword methods: random-replacement based (e.g., tweaking tail [21]) and password-model based (see Sec. III-E).

Now U_i's account record in S can be represented as (ID_i, SW_i), where $\mathrm{SW}_i = (\mathrm{sw}_{i,1}, \mathrm{sw}_{i,2}, \cdots, \mathrm{sw}_{i,k})$. Exactly one of these k sweetwords, denoted by $\mathrm{sw}_{i,j}$, equals U_i's password PW_i. Let C_i denote the correct index of U_i's password in the sweetword list SW_i, and thus $C_i = j$. The k sweetwords on S shall be hashed, with salting in a per-user or even per-sweetword manner. The record (ID_i, C_i) is kept on the honeychecker which is a separate, hardened computer system of minimalist design. It may be placed in different administrative domains, runs different operating system, software stacks, security mechanisms and so on, and ensures distributed security [7], [21]. The honeychecker is not publicly accessible, and it only interacts with S by using a "dedicated and/or encrypted and authenticated" [21] communication channel.

When U_i logs in with (ID_i, PW_i^*), S first looks up the list SW_i and sees whether there is one element (with index C_i^*) that matches PW_i^*. If not, the login is rejected. Otherwise, S submits a command $\mathsf{Check}(\mathrm{ID}_i, C_i^*)$ to the honeychecker. If C_i^*=C_i, then the honeychecker signals to S to accept U_i. Otherwise, it suggests that a login with honeyword is attempted, and an alarm is raised to S. Depending on the alarm policy, S may take an appropriate action, such as: 1) accept the login but on a honeypot system, and more stringently monitor the user's activities; 2) if the number of honeyword logins against U_i's account exceeds a pre-defined threshold $\mathcal{T}_1$ (e.g., 3), lock out U_i's account until the user resets a new password; or 3) shut down the computer system and require all users to reset new passwords, if all users' total honeyword login attempts exceeds a pre-defined threshold $\mathcal{T}_2$. The value of $\mathcal{T}_2$ depends on the system's risk analysis and is out of our scope. Since the system has to balance honeyword-distinguishing attacks and DoS attacks, $\mathcal{T}_2$ shall not be too small or too large, and without loss of generality, we set $\mathcal{T}_2$=10^4.

Honeyword distinguishing attacker. As mentioned earlier, the most essential security goal of any honeyword method is to produce a set of k-1 honeywords for a given user U_i's account such that they shall be *in*distinguishable from U_i's real password PW_i. This goal corresponds to the honeyword distinguishing attacker $\mathcal{A}$ as shown in Fig. 1, who aims to tell the real password apart from the k-1 honeywords associated with U_i's account by using S as a querying oracle. $\mathcal{A}$'s honeyword login attempts will be detected by the honeychecker, and if the number of such attempts against U_i's account 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 raise the system-wide alarm if her login attempts exceed the threshold $\mathcal{T}_2$ (e.g., 10^4). Thus, *$\mathcal{A}$'s honeyword login attempts shall be as few as possible*.

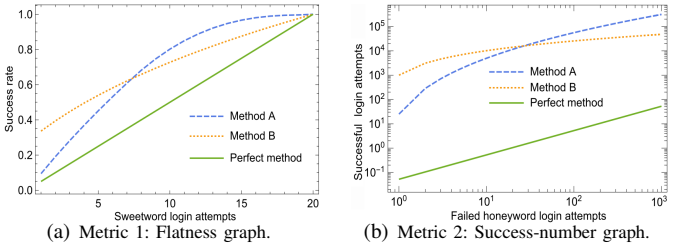
We assume that $\mathcal{A}$ has somehow already got access to the server S's password hash file, knows the algorithm under which the honeywords are generated and hashed, and is armed with all the publicly available information (e.g., various publicly leaked password datasets and the target site's password policy). We call this attacker a type-$\mathcal{A}_1$ attacker. As said earlier, $\mathcal{A}$ may also obtain the victim U_i's PII. We call this advanced attacker a type-$\mathcal{A}_2$ attacker. These assumptions about $\mathcal{A}$'s capabilities are indeed realistic, yet they are often only implicitly made (or missed) in previous studies [7], [14], [21].

Other attackers. As discussed in [21], other valid threats facing honeywords include attacking the honeychecker system, denial-of-service (DoS) attacks that game the honeyword system and intersection attacks where an attacker exploits user passwords reused across different systems. Since these attacks have little relevance to the strength of a honeyword-generation method, and thus they are beyond our scope.

C. Evaluation metrics

Juels and Rivest [21] proposed a notion of ϵ-flat to measure the security of a honeyword generation method. ϵ-flat denotes the maximum success rate ϵ that, when given U_i's k sweetwords, the distinguishing attacker $\mathcal{A}$ can gain by submitting only *one* online guess to S. However, this metric is inadequate for measuring a method's security level when $\mathcal{A}$ is allowed to make more than one online guess per user (e.g., when $\mathcal{T}_1 > 1$). In addition, ϵ-flat falls short of reflecting the system's "low-hanging fruits" produced by a method, i.e. these most vulnerable honeywords that can be easily distinguished.

Accordingly, we propose two new metrics: *flatness graph* and *success-number graph*. These two metrics (see Fig. 2)



(a) Metric 1: Flatness graph. (b) Metric 2: Success-number graph.

Fig. 2. Two metrics for measuring the resistance of a honeyword generation method against the distinguishing attacker. Here methods A and B are conceptual. The closer to the perfect line, the better a method will be.

measure a method's resistance against the honeyword distinguishing attacker in the *average* and *worse-case* point of view.

Flatness graph plots the probability of distinguishing the real password vs. the number of sweetword login attempts per user. As shown in Fig. 2(a), a point (x, y) on a curve indicates that the real password is guessed with a probability y in the first x attempts, where $x \leq k$. In reality, it also requires that $x \leq \mathcal{T}_1$ (e.g., $\mathcal{T}_1$=1 or 3). Clearly, the data point (x=1, y) on our flatness curve corresponds to the ϵ-flat metric introduced in [21], i.e., $\epsilon = y|_{x=1}$. A flatness graph provides a view of the *average* resistance against a distinguishing attacker with varied guess numbers per user all the way to k.

Success-number graph measures to what extent a method will produce vulnerable "low-hanging fruits" for $\mathcal{A}$. This graph plots the total number of successful login attempts (i.e., login with a real password) vs. the total number of failed login attempts (i.e., login with a honeyword). A point (x, y) on a curve in Fig. 2(b) indicates that y real passwords are successfully distinguished from the system before the x-th honeyword login attempt occurs, where $x \leq \mathcal{T}_2$ (e.g., $\mathcal{T}_2$=10^4). A $\frac{1}{k}$-flat method is perfect, since it produces *no* "low-hanging fruits" and each of the k sweetwords per user is of the *same* probability to be a real password.

III. Trawling Guessing Attacks

The stated security goal for Juels-Rivest's four primary honeyword-generation methods is that, when given U_i's k sweetwords, the distinguishing attacker $\mathcal{A}$ can gain a maximum success rate about $\frac{1}{k}$ by making an online query to S (see Table 1 of [21]). However, by using large-scale real-world datasets as listed in Table I, we now show that these methods all fall short of its intended security goal. We first provide a review of Juels-Rivest's methods, and then show that their methods fail to achieve the expected security (i.e., $\frac{1}{k}$-flat) under the weak assumptions of the attacker (i.e., a type-$\mathcal{A}_1$ attacker).

A. Review of Juels-Rivest's methods

Juels-Rivest's four main methods (see Table 1 of [21]) are all based on the random-replacement approach: each honeyword is produced by randomly replace parts (or whole) of the real password. Particularly, the first three are real-password related ones, i.e., the $k-1$ honeywords depend on the structure of the real password PW_i.

Tweaking by tail. Their first method is to "tweak" 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. Juels and Rivest [21] mainly discussed the "tweaking by tail" version, and we also focus on this version. Each character in the last t positions of PW_i is replaced by a randomly-chosen character of the same type: a digit is replaced by a digit, a letter by a letter, and a special character by a special character. For instance, if PW_i is `trustno1` and $t = 3$, then the sweetword list SW_i might be `trustyp0`, `trustmi7`, `trustno1`, `trustme5`, etc.

Modeling syntax. Their second method, inspired by [4], is to parse PW_i into tokens of consecutive characters of the same type and extract the corresponding syntax, and then honeywords are generated by replacing tokens with randomly selected values that match the syntax. For instance, the syntax $\mathsf{L}_7\mathsf{D}_1$ can be abstracted from `trustno1`, and then honeywords, such as `dfdphus3`, `letmein4` and `kebrton9` can be generated.

Hybrid. Their third method is to combine the tweaking-tail and modelling syntax methods, aiming to gain advantages from both methods. More specially, when given U_i's password PW_i, the hybrid method first employs the modeling syntax approach to extract PW_i's syntax and produces a seed sweetwords (including PW_i), and then generates $b-1$ honeywords by tweaking each seed sweetword, where $a \cdot b \geq k$. Finally, $k-1$ honeywords except for PW_i are randomly selected from these $a \cdot b - 1$ honeywords. It is recommended that $a \approx b \approx \sqrt{k}$. For instance, assume k=20, and a=4, b=5; the syntax $\mathsf{L}_7\mathsf{D}_1$ can be abstracted from `trustno1`, and 3 seed sweetwords `dfdphus3`, `letmein4` and `kebrton9` are generated. Then, 4 new honeywords are produced for *each* of the 4 seed sweetwords, resulting in 20 sweetwords for U_i.

Simple model. Their fourth method is to use real password samples as an aid to generate honeywords (see Appendix of [21]). First, a list L is built by combining large-scale real-life passwords and 8% random passwords of varying lengths. Then, a random element $w = w_1 w_2 \cdots w_d$ of length d is picked from L, and a honeyword c with char sequence $w_1 c_2 \cdots c_d$ is generated heuristically. More specifically, for $j = 2, 3, \cdots d$,

- With probability 10%, replace w by a random element in L, and then set $c_j = w_j$;
- With probability 40%, replace w by a random element in L satisfying $c_{j-1} = w_{j-1}$, and set $c_j = w_j$;
- With probability 50%, set $c_j = w_j$.

For fair evaluation, when implementing Juels-Rivest's methods, we strictly follow the specifications in [21].

Remark 1. We note that in *a passing comment*, Juels and Rivest mention the possibility of using probabilistic password models (e.g., Weir et al.'s PCFG [32]) to build honeywords: "see Weir et al. [32] for a presentation of an interesting alternative model for passwords, based on probabilistic context-free grammars" [21]. However, due to the Zipf-distribution nature of passwords [28], generating decoy passwords (according to a probabilistic password model or not) that are equally probable to the user's real password is *inherently impossible* (see Sec. VI). Moreover, each password model has its own inherent weaknesses. Other challenges like sparsity also arise. They all make the use of probabilistic models *not* straightforward but rather challenging, Juels and Rivest explicitly leave it as an open question (see Section 9 of [21]): "can the password

models underlying cracking algorithms (e.g., [32]) be easily adapted for use?" In Sec. III-E, we will provide a negative answer to this question.

Remark 2. Also note that, besides the above four methods, there is another legacy-UI method called "Chaffing with tough nuts". It needs to be used together with other methods, and aims to generate some tough honeywords that are much harder to crack to render $\mathcal{A}$'s offline guessing work more challenging. However, as pointed out by Erguler [14], the attacker $\mathcal{A}$ cares cost-effectiveness: if the cracking time exceeds her acceptable time-threshold, $\mathcal{A}$ would stop cracking these tough nuts and simply avoid using these "tough nuts", because common user rarely use "tough nuts". In this regard, introducing "tough nuts" as honeywords would make the number of a user's effective honeywords less than k-1 and thus actually decreases security. Hence, we do not consider it.

B. Two attacking strategies

We now present two simple yet practical honeyword distinguishing strategies (with varying tunings) against Juels-Rivest's methods. They work on real-world password datasets, inherently different from the existing heuristic strategy (see [7], [14], [21]) that is mainly based on some *specific* counter-example passwords. Essentially, they make use of the fact that user-chosen passwords follow the Zipf's law [28], and ranks the sweetwords based on some known probability distribution, either normalized within a user or not. The probability distribution can be calculated from a leaked password dataset, or based on a probabilistic password model such as Markov [24].

For fair evaluation, here we consider the attacker capabilities of $\mathcal{A}_1$, i.e., a basic attacker (see Sec. II-B). In brief, $\mathcal{A}_1$ has got the password hash file, knows all cryptographic algorithms and public info, but does not exploit user PII. Such an assumption of attacker capabilities is in line with Juels-Rivest's work [21] as well as the other previous studies [7], [14].

Top-PW attack. This attack is simple yet effective. The distinguishing attacker $\mathcal{A}$ is faced with a password file F of n lists $\{\mathrm{SW}_1, \mathrm{SW}_2, \cdots, \mathrm{SW}_n\}$, where each list consists of a set k of distinct sweetwords. Given such a multi-set of $n \cdot k$ sweetwords, $\mathcal{A}$'s aim is to find as many real passwords as possible before making $\mathcal{T}_2$ (e.g., 10^4) failed honeyword login attempts. Note that $\mathcal{A}$ can only make at most $\mathcal{T}_1$ (e.g., 1, 3, or 10) honeyword logins against each account. $\mathcal{A}$ simply tries these $n \cdot k$ sweetwords in decreasing order of probability, where the probability of each sweetword $\mathrm{sw}_{i,j}$ ($1 \le i \le n$ and $1 \le j \le k$) comes directly from a known probability distribution $P_{\mathcal{D}}$ (e.g., a leaked dataset $\mathcal{D}$ like the 32 million Rockyou) as follows:

Step 1. $\forall \mathrm{sw}_{i,j} \in F$, if $\mathrm{sw}_{i,j} \in \mathcal{D}$, set $\Pr(\mathrm{sw}_{i,j}) = P_{\mathcal{D}}(\mathrm{sw}_{i,j})$;
Step 2. $\forall \mathrm{sw}_{i,j} \in F$, if $\mathrm{sw}_{i,j} \notin \mathcal{D}$, set $\Pr(\mathrm{sw}_{i,j}) = 0$.

We call $P_{\mathcal{D}}(\cdot)$ the List password model, where $\forall x \in \mathcal{D}$, $P_{\mathcal{D}}(x) = \frac{\mathtt{Count}(x)}{|\mathcal{D}|}$, where $\mathtt{Count}(x)$ means the occurrence number of password x and $|\mathcal{D}|$ is the size of the multi-set $\mathcal{D}$. We empirically evaluate this attack against the four methods in [21]. More specifically, we randomly split the Dodonew dataset into two parts of equal size, resulting in: 8.129 million Dodonew-tr and 8.129 million Dodonew-ts. For each method in [21], $k-1$ honeywords are produced for each of the password in Dodonew-ts, while $P_{\mathcal{D}}$ of the Top-PW attack comes from Dodonew-tr. As shown in Figs. 3(a)∼3(d), the Top-PW attack tells apart at least 615,664 passwords from the 8,129,445 Dodonew-ts accounts protected by any of the four methods in [21]. However, according to Juels-Rivest's original security goal, an attacker should distinguish only about 526(=$\mathcal{T}_2/(k-1)$) real passwords from Dodonew-ts. That is, *in terms of the success-number metric, their methods are weaker than the claimed security by a factor of at least 1170.*

Algorithm 1: Distinguish the real passwords from a given password file F that is consisted of n lists of sweetwords.

Input: n sweetword lists $\{\mathrm{SW}_1, \mathrm{SW}_2, \cdots, \mathrm{SW}_n\}$, one list per user; the threshold $\mathcal{T}_1$ of honeyword login attempts per user; the threshold $\mathcal{T}_2$ of honeyword login attempts for the system.
Output: A vector $\mathcal{V}$ shows the success-number graph.

```
Initialize the vector V to be ∅;
Initialize the priority queue crackQueue to be ∅; /* crackQueue is ranked by the conditional probability of sweetwords. */
numFailure = 0, numSucess = 0;
for i = 1 to n do
    (p, sw) = getSweetword(SW_i); /* according to the specified attacking strategy, get the sweetword sw with the highest priority p normalized among all the un-attempted sweetwords in SW_i. */
    crackQueue.Insert((p, SW_i, sw));
while numFailure < T_2 and !crackQueue.empty() do
    (p, SW_j, sw) = crackQueue.getFirst();
    crackQueue.removeFirst();
    if SW_j.numFailure < T_1 then
        success=Login(SW_j.id, sw);/*use SW_j.id as user name and sw as password to login as user U_j. */
        if success then
            numSuccess++;
        else
            SW_j.numFailure++, numFailure++;
            V.Insert((numFailure, numSuccess));
            (p, sw) = getSweetword(SW_j);
            crackQueue.Insert((p, SW_j, sw));
return V;
```

We also employ the Top-PW attacking strategy to evaluate the flatness of the four methods in [21]. In brief, when given a list SW_i of k sweetwords for user U_i, $\mathcal{A}$ simply tries these k sweetwords in decreasing order of probability, where each sweetword's probability comes directly from a known probability distribution $P_{\mathcal{D}}$ as above. Fig. 3(e) shows that all methods are 0.35^+-flat. In other words, there is over 35% of success rate in telling part the real password from the k=20 sweetwords with just *one* guess. See more details in Sec. III-D. Evidently, there is a large gap between these four methods and the expected (perfect) method.

Though this attack is effective, it is subject to an obvious defect: when identifying the next sweetword $\mathrm{sw}_{i,j}$ to be guessed, this identifying process bears little relevance to the nature of the underlying sweetword list to which $\mathrm{sw}_{i,j}$ belongs. For example, assume there are two sweetword lists left: SW_i and SW_{i+1}, where (1) $\Pr(\mathrm{sw}_{i,j})$=0.030 and all the other 19 sweetwords in SW_i are with a equal probability 0.029; (2) $\Pr(\mathrm{sw}_{i+1,j})$=0.028, and all the other 19 sweetwords in SW_{i+1} are with a equal probability 0.001. Then, which sweetword shall be tried next, $\mathrm{sw}_{i,j}$ or $\mathrm{sw}_{i+1,j}$? It is more likely that $\mathrm{sw}_{i+1,j}$ will be the real password for SW_{i+1} than $\mathrm{sw}_{i,j}$ for SW_i. Thus, it is better to first try $\mathrm{sw}_{i+1,j}$ against user U_{i+1}. This intuition gives rise to the following more effective attacking strategy.

Normalized top-PW attack. The essential difference between this strategy and the above one is that, $\mathcal{A}$ now tries these $n \cdot k$ sweetwords from lists $\{\mathrm{SW}_1, \mathrm{SW}_2, \cdots, \mathrm{SW}_n\}$ in decreasing order of *normalized* probability. More specifically, the probability of each sweetword $\mathrm{sw}_{i,j}$ ($1 \le i \le n$ and $1 \le j \le k$) comes directly from a known password distribution $\mathcal{D}$ (i.e., using the List password model $P_{\mathcal{D}}(\cdot)$), but it has been normalized among each user's own k sweetwords:

Step 1. $\forall \mathrm{sw}_{i,j} \in \mathrm{SW}_i$, if $\mathrm{sw}_{i,j} \in \mathcal{D}$, set $\Pr(\mathrm{sw}_{i,j}) = P_{\mathcal{D}}(\mathrm{sw}_{i,j})$;

Step 2. $\forall \mathrm{sw}_{i,j} \in \mathrm{SW}_i$, if $\mathrm{sw}_{i,j} \notin \mathcal{D}$, set $\Pr(\mathrm{sw}_{i,j}) = 0$;

Step 3. $\forall \mathrm{sw}_{i,j} \in \mathrm{SW}_i$, set $\Pr(\mathrm{sw}_{i,j}) = \Pr(\mathrm{sw}_{i,j}) / \sum_{t=1}^{k} \Pr(\mathrm{sw}_{i,t})$.

If the system allows more than one honeyword login attempt (i.e., $\mathcal{T}_1 > 1$), after any one of the sweetwords in SW_i has been attempted, the probability of *all the other unattempted sweetwords* in SW_i shall be normalized:

Step 3'. Let $\mathcal{I}$ denote the set of subscripts of all the sweetwords in SW_i that have *already* been used in login attempts. $\forall \mathrm{sw}_{i,j} \in \mathrm{SW}_i \wedge j \notin \mathcal{I}$, set $\Pr(\mathrm{sw}_{i,j}) = \Pr(\mathrm{sw}_{i,j}) / \sum_{t \notin \mathcal{I}} \Pr(\mathrm{sw}_{i,t})$.

After the probability of each *un*attempted sweetwords in SW_i has been normalized, one can sort the sweet list SW_i and identify the one with the highest priority. This corresponds to an instantiation of the function `getSweetword`(SW_i) in Line 5 of Algorithm 1. Once it has been instantiated, a run of Algorithm 1 will produce the success-number graph. Similarly, after the Steps 1∼3, every sweetword in SW_i is with a (normalized) probability, and thus the function `getSweetword`(SW_i) in Line 5 of Algorithm 2 is instantiated. Finally, a run of Algorithm 2 produces the flatness graph.

Since password distributions in reality differ greatly from each other (see concrete examples in Fig. 3 of [30]), many sweetwords in the password file F do not appear in the distribution $\mathcal{D}$, and they will be assigned a probability 0. This causes great inaccuracies in sweetword ranking. For instance, when using Dodonew-tr as the training set (i.e., to be $\mathcal{D}$) and Dodonew-ts as the test set, over 74% sweetwords in the file F (which is generated from Dodonew-ts) will be assigned a probability 0 according to Step 2. Crucially, one should not think that things that one has not yet seen are of probability 0. This is called sparsity issue in machine learning, and it can be addressed by applying the techniques of smoothing. A number of smoothing methods have been proposed for language modeling, such as Laplace and Good-Turing, and they have been widely used in password studies [24], [30]. However, they cannot be readily applied to our settings where the majority of sweetwords do not appear in the training set. We devise an "+1" smoothing method:

Step 2'. $\forall \mathrm{sw}_{i,j} \in \mathrm{SW}_i$, if $\mathrm{sw}_{i,j} \notin \mathcal{D}$, set $\Pr(\mathrm{sw}_{i,j}) = \frac{1}{|\mathcal{D}|+1}$.

We have experimented with the Laplace, Good-Turing and "+1" smoothing methods, and found our "+1" method though simple yet most effective. Thus, we prefer the "+1" method. Note that, "+1" method assigns the *same*, constant value $\frac{1}{|\mathcal{D}|+1}$ to all sweetwords that have not appeared, and thus it is mainly effective for ranking *popular* sweetwords, mainly suitable for attacking Juels-Rivest's four methods. When the main goal is to rank *un*popular sweetwords, other smoothing methods shall be designed. In Sec. III-E, we will devise a new strategy for attacking *un*popular sweetwords.

Algorithm 2: Distinguish the real password from *every list of sweetwords* in a given password file F.

Input: n sweetword lists $\{\mathrm{SW}_1, \mathrm{SW}_2, ..., \mathrm{SW}_n\}$, one list per user.
Output: A vector $\mathcal{V}$ shows the flatness graph.

```
1  Initialize the vector V to be empty;
2  for i = 1 to n do
3      r = 0;/* r counts the attempt num before a success for SW_i.*/
4      while !SW_i.isEmpty() do
5          (p, sw) = getSweetword(SW_i);  /* according to the
           specified attacking strategy, get sweetword sw with the highest
           priority p among all un-attempted sweetwords in SW_i. */
6          success=Login(SW_i.id, sw);/*use SW_i.id as user name
           and sw as password to login as user U_i. */
7          SW_i.deleteSweetword(sw);
8          r++; /* the attempt num for SW_i increases by one. */
9          if success then
10             break   /* if succeed, move to the list SW_{i+1}. */
11     V.Insert(r);
12 return V
```

Remark 3. It is worth noting that, the key difference between the above two attacking strategies, i.e. the (additional) Step 3 of the Normalized top-PW, will have a *large* effect on the success-number graph. However, this difference has *no* effect on the flat graph, because all of each user's k sweetwords will be attempted and the Step 3 of the "normalized Top-PW" strategy does *not* change the relative rankings *within* each sweetword list. Both strategies (and the third one "Norm PW-model" in Sec. III-E) do not mean that "if a password appeared in a breach list $\mathcal{D}$, and appears in the list of honeywords, it is the actual password". Generally, there are j ($1 \le j \le k$) sweetwords appear in $\mathcal{D}$ and $k - j$ sweetwords don't, and the key difficulty lies in how best to rank these k sweetwords. This requires creative efforts: new normalization and smoothing techniques, and retooling password models to eliminate sweetword ties which have the same probability.

"Using a blacklist" is ineffective against all our proposed attacks, because: (1) Each service has vastly different password distributions—As shown in Fig. 3 of [30], every two services share less than 40% of passwords, and thus no pre-constructed blacklist will block the majority of popular passwords; (2) Users tend to circumvent the blacklist [8], [28], and new popular passwords will arise: if `password` is blocked, `password1` will arise; if `password1` is blocked, `password123` (and `p@ssword1`) will arise; (3) Blacklist is inherently unable to recognize/block PII-based passwords, and thus is ineffective against the Type-$\mathcal{A}_2$ attacker.

Remark 4. Essentially, our two honeyword attacking strategies are effective in distinguishing popular sweetwords from *un*popular ones. Thus, when more popular sweetwords are more likely to be users' passwords, our attacking strategies will be effective. This indicates that they are particularly suitable for attacking *non*-password-model based methods (e.g., Juels-Rivest's four methods) which generate a set of $k - 1$ honeywords that are generally *less probable* than the user's original password. When password-model based methods are in place, the main goal is to rank *un*popular sweetwords, and other attacking strategies shall be designed. In Sec. III-E, we will show how to revise the "Normalized top-PW attack" strategy for attacking password-model based methods.

C. Experimental setups

When evaluating a honeyword method, the success-number graph is resulted from a run of Algorithm 1, while the flatness graph from a run of Algorithm 2. The differences in each evaluation lie in how the function `getSweetword(`SW_i`)` is instantiated: the attacking strategy (i.e., Top-PW, Norm top-PW, or Norm PW-model in Sec. IV), the training set and the test set. For each attacking strategy, there are 7 different password models (i.e., List, PCFG [32], Markov [24], their targeted versions [30], and no-training-set) for possible choice.

In most of our experiments, we use the first half of a password dataset for training (e.g., 8.129 million Dodonew-tr), and the second half for testing (e.g., 8.129 million Dodonew-ts). Now we explain why. On the one hand, the effectiveness of a machine-learning-based distinguishing algorithm depends on two factors: the algorithm itself and the training sets used. By randomly dividing a dataset into equally two parts, and using part-1 for training and part-2 for testing, we can preclude the impacts of training sets when evaluating an algorithm. On the other hand, as unending lists of real-world password lists have been disclosed (see [18], [20], [22]), the attacker can constantly improve her training set to make it as close as possible to the test set. For instance, the target system's password distribution can be largely approximated by a leaked site with the same language, service type, password policy, etc. Actually, many sites (e.g., Yahoo [18], Phpbb and Anthem [26]) have leaked their user passwords more than once. Thus, we argue that when evaluating a honeyword generation method, it is desirable to employ a powerful yet realistic attacker and *train the attacker on a training set close to the test set*.

This "half-half" practice does not violate the machine-learning principle that the training set and test set shall be different. It enables the attacker knows a large portion of the knowledge of the target system's password distribution, but not exactly the full knowledge. The underlying reason is that, according to the scale-free nature of Zipf's law in passwords [28], a non-negligible portion of passwords in "the first half of dataset for training" will not appear in "the second half for testing", making the training set and test set similar but not the same. For instance, among the 5.59 million distinct passwords in the test set Dodonew-ts, 4.54 million (81.22%) do not appear in the training set Dodonew-tr. Actually, this practice is also quite routine in password research (see [9], [30], [32]), but we for the first time explain why it is acceptable.

Besides experiments where the training set and test set stem from the same original distribution, we also investigate the impacts of varied training sets which come from different web services in Sec. III-E. In addition, our non-train-set model (see Sec. IV-A) does not employ any training set at all. In our figures, generally, the lower the line, the better the corresponding honeyword-method will be; Whenever a perfect method line is presented, the closer to the perfect line, the better the corresponding honeyword-method will be.

D. Our basic, trawling guessing attacks

In the above two attacking strategies, we instantiate the probabilities of sweetwords for a given password file by using the List password model $P_{\mathcal{D}}(\cdot)$ as defined in Sec. III-B: $\forall x \in \mathcal{D}$, $P_{\mathcal{D}}(x) = \frac{\mathtt{Count}(x)}{|\mathcal{D}|}$). That is, the probability of a sweetword *directly* comes from the training set $\mathcal{D}$. We call it the basic, trawling guessing attack. In Sec. III-E, we will instantiate the probabilities of sweetwords by using much more sophisticated password models like PCFG and Markov.

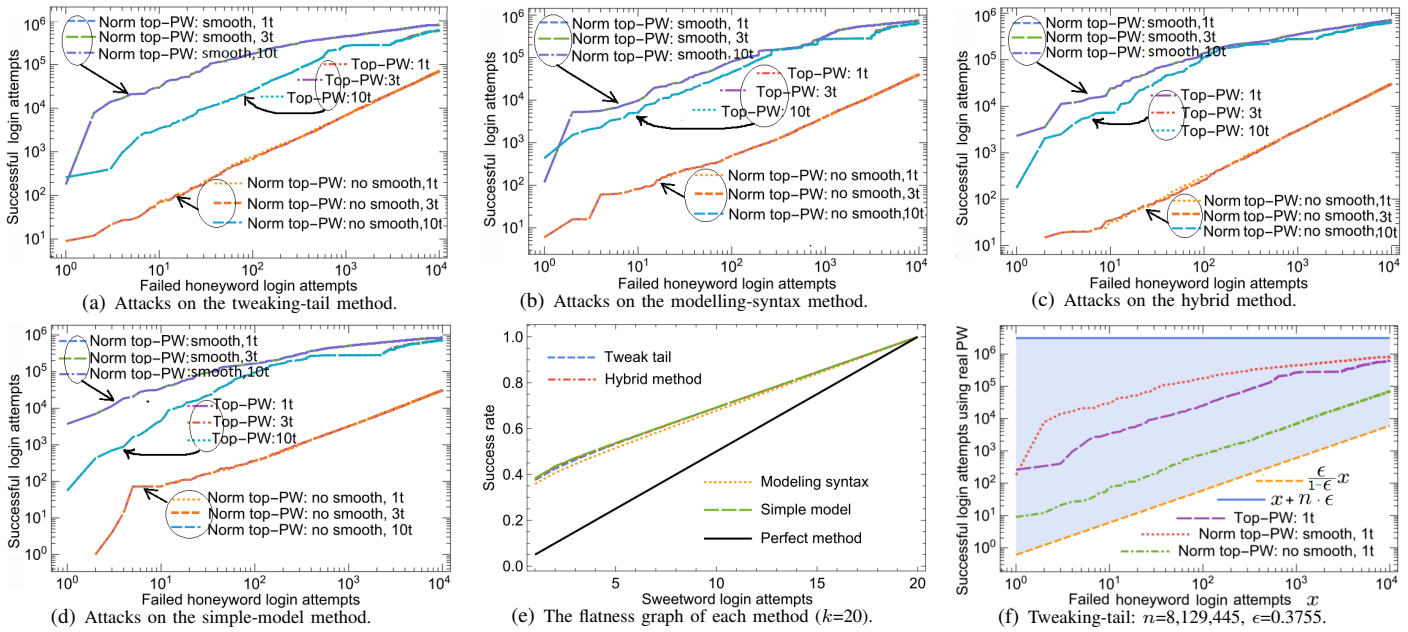
Figs. 3(a)∼3(d) show that, in terms of the success-number graph, the "norm Top-PW" attack strategy with smoothing performs significantly better than the no-smoothing version. Generally, it also performs much better than the "Top-PW" strategy, especially when the login number allowed is small. *This suggests the critical role of smoothing in attacking honeywords*. When trained on Dodonew-tr, it can tell at least 710930 (8.75%) real passwords apart from the 8,129,445 Dodonew-ts accounts protected by any of the 4 methods in [21]. This indicates that, in terms of the success-number metric, there are over 1352=(710930/($\mathcal{T}_2$/(k-1))) times of *underestimation* of the vulnerabilities in Juels-Rivest's methods.

As said earlier, when evaluating the flatness security goal of the methods in [21], the two attacking strategies (see Sec. III-B) will essentially try the *same* sequences of sweetwords for each user account. Hence, they will produce the same flatness graph as shown in Fig. 3(e). Our results show that all 4 methods provide very similar security levels of flatness, with *the modeling syntax being slightly better*. Their similarity holds in all our later experiments, and due to space constraints, hereafter we only take the tweaking-tail method as an example.

As shown in Figs. 3(a)∼3(d), no matter each user account can be attacked 1, 3, or 10 times, both attacking strategies perform rather stably. The underlying reasons are that: (1) when an account can be attacked just once (i.e., $\mathcal{T}_1$=1), both strategies can successfully identify at least 615,664 real passwords from Dodonew-ts against every method in [21]; and (2) When an account can be attacked more than once (i.e., $\mathcal{T}_1$>1), there are no more than $\mathcal{T}_2$ *un*-distinguished user accounts involved in the attack, and thus $\mathcal{A}$ will crack at most *an additional* $\mathcal{T}_2$ real passwords as compared to $\mathcal{T}_1$=1; and (3) 615,664$\gg\mathcal{T}_2$, thus 615,664+$\mathcal{T}_2\approx$615,664. Therefore, from then on we only consider the case where $\mathcal{T}_1$=1 when evaluating the success-number graphs. Note that, *the setting of $\mathcal{T}_1$=1 has no relevance to our evaluation of the flatness graph where k login attempts per user are always allowed.*

Recall that the success-number metric measures a method's strength against the distinguishing attacker $\mathcal{A}$ in the *worse-case* point of view, i.e., $\mathcal{A}$ first attacks the weakest user accounts. Take the case of "Top-PW: 1t" in Fig. 3(a) for example. Since `123456` is the top-1 password (1.44%) in Dodonew-tr, $\mathcal{A}$ will first use `123456` as the guess to test against all the 8,129,445 Dodonew-ts accounts. One might think that, 1.44% · 8,129,445 ≈ 110K successful logins shall occur before the 1st failed login occurs. However, this is not true, because `123456` can also be a *honeyword* for user accounts that are with a real password of the pattern `123xxx` (e.g., `123123`). Thus, $\mathcal{A}$'s login with `123456` may fail for such accounts before reaching 110K successful logins. Fig. 3(a) shows that, to reach 110K successful logins, $\mathcal{A}$ fails about 100 times when using the "Norm Top-PW with smoothing" strategy, and about 1000 times when using the "Top-PW" strategy.

In the above, we mainly used the Dodonew dataset to evaluate Juels-Rives's methods. The evaluation using the other 9 datasets show similar results, and due to space constraints, they are omitted here. The evaluation figures of three re-



(a) Attacks on the tweaking-tail method. (b) Attacks on the modelling-syntax method. (c) Attacks on the hybrid method.

(d) Attacks on the simple-model method. (e) The flatness graph of each method (k=20). (f) Tweaking-tail: n=8,129,445, ϵ=0.3755.

Fig. 3. Experiment results for attacking the four methods in [21] in terms of the success-number and flatness metrics. Each method is evaluated by two attacking strategies with various tunings, trained on 50% of Dodonew (i.e., Dodonew-tr) and tested on the remaining 50% (i.e., Dodonew-ts). Whenever a perfect method line is presented, the closer to the perfect line, the better the corresponding honeyword method will be. Sub-figures (a)∼(d) show that the "norm top-PW" attacking strategy with smoothing can distinguish 711K$^+$ real PWs against every method when allowed $\mathcal{T}_2$=10^4 honeyword logins, where $1t$ means $\mathcal{T}_1$=1, $3t$ means $\mathcal{T}_1$=3 and so on; Sub-figure (e) reveals that all 4 methods are 0.35$^+$-flat, over 7 times weaker than expected in [21]. Sub-figure (f) exemplifies the correlations between the flatness and success-number metrics: (1) the upper-bound of success-number is $x+n\cdot\epsilon$, which means before the x-th failed login occurs, all the total Dododnew-ts accounts (i.e., n=8,129,445) have already been tried (since $x \ll n*\eta$, then $x+n*\eta \approx n*\eta$, being a horizontal line); and (2) the lower-bound is $\frac{\epsilon}{1-\epsilon}\cdot x$, which means before the x-th failed login occurs, $\mathcal{A}$ has at least tried $\frac{1}{1-\epsilon}\cdot x$ accounts, and her success number is $\epsilon\cdot(\frac{1}{1-\epsilon}\cdot x)$.

TABLE IV. SUCCESS-NUMBER INFORMATION (%) ABOUT EACH HONEYWORD METHOD, EVALUATED UNDER THE (TYPE-$\mathcal{A}_1$) "NORM TOP-PW: SMOOTH, 1T" STRATEGY AND $\mathcal{T}_2 = 10^4$.†

	Tweak-tail	Model-syntax	Hybrid	Simple model
Tianya	14.41%	13.04%	14.90%	**5.81%**
Dodonew	10.10%	9.06%	10.46%	**8.75%**
CSDN	18.78%	**15.75%**	18.39%	16.32%
12306	9.32%	**7.88%**	9.17%	9.51%
Rockyou	21.63%	7.35%	14.01%	**2.41%**
000webhost	9.56%	14.33%	16.86%	**4.56%**
ClixSense	16.87%	**5.27%**	9.52%	6.08%
Yahoo	24.25%	**7.61%**	13.81%	16.84%
Rootkit	20.39%	**12.72%**	17.82%	19.57%
QNB	20.99%	20.85%	20.97%	**20.48%**
Average	16.63%	11.39%	14.59%	**11.03%**

†A value in bold means the corresponding method perform best among four methods, while a value with background color means the worst.

TABLE V. ϵ-FLAT INFO ABOUT EACH HONEYWORD METHOD.†

	Tweak-tail	Model-syntax	Hybrid	Simple model
Tianya	**0.4368**	0.4400	0.4580	0.4463
Dodonew	0.3755	**0.3582**	0.3796	0.3828
CSDN	0.3664	**0.3437**	0.3716	0.3978
12306	0.1309	**0.1177**	0.1287	0.1327
Rockyou	0.5498	**0.4831**	0.5334	0.5035
000webhost	0.3550	0.3587	0.3594	**0.3541**
ClixSense	0.3055	**0.2221**	0.2758	0.2943
Yahoo	0.2785	**0.2080**	0.2527	0.2661
Rootkit	0.2293	**0.1636**	0.2052	0.2210
QNB	0.2348	0.2342	0.2355	**0.2313**
Average	0.3262	**0.2929**	0.3200	0.3230

†Both our attacking strategies in Sec. III-B would produce the same flatness graph. The experimental setups and the meaning of the bold/background-color emphasis are the same with that of Table IV.

maining million-sized datasets (i.e., Tianya, Rockyou and 000webhost) can be found in the companion site. Tables IV and V summarize all the evaluation results for our ten datasets. As said earlier, the choices of training and test sets follow the "half-half" practice. When allowing 10000 failed attempts, 11.03%∼16.63% accounts of the ten test sets can be successfully guessed; Tweak-tail provides the worst ϵ-flatness (i.e., 0.3262), even the best method Model-syntax are 0.2929-flat: users' real passwords can be distinguished with a success rate of 29.29% by a trawling guessing attacker, but not the expected 5%, with just *one* guess (when k=20 as recommended). No method always performs the best in both metrics, yet in general there does show a hierarchy: Model-syntax > Simple model ≈ Hybrid > Tweak-tail.

Remark 5. Why our experiments are so effective? It has been discussed in Remark 4 that, our "Normalized top-PW" attacking strategy is essentially effective in distinguishing popular sweetwords from *un*popular ones. Thus, it is particularly suitable for attacking Juels-Rivest's four random-replacement methods which generate a set of k-1 honeywords that are generally *less probable* than the user's original password. For instance, Table VI shows that the typical most popular passwords in Dodonew-ts was *rarely* (<0.01%) generated as a honeyword by any method. This means that, once these popular passwords appear in a user's sweetword list, they are probably a real password, and our algorithm can identify them.

E. Further evaluations

Varying training sets. In the above, we have evaluated Juels-Rivest's four primary methods under two attacking strategies with various tunings. In all experiments, the training set and the test set stem from the same original distribution, and they would be of quite similar distributions. One may wonder

TABLE VI. AN ILLUSTRATION OF PERCENTAGES OF POPULAR PASSWORDS THAT APPEAR IN A SWEETWORD LIST (EITHER AS A REAL PASSWORD OR AS A HONEYWORD).†

Top PWs	Dodonew-ts	Generated as honeywords for Dodonew-ts			
	Real PW	Tweak-tail	Model-syntax	Hybrid	Simple model
123456	1.4437%	0.0057%	0.0000%	0.0016%	0.0003%
a123456	0.3818%	0.0013%	0.0036%	0.0003%	0.0002%
123456789	0.3199%	0.0004%	0.0000%	0.0008%	0.0014%
111111	0.2273%	0.0004%	0.0033%	0.0002%	0.0005%
5201314	0.1898%	0.0002%	0.0000%	0.0010%	0.0035%
123123	0.1741%	0.0299%	0.0000%	0.0049%	0.0000%
a321654	0.1670%	0.0001%	0.0003%	0.0000%	0.0008%
woaini	0.0383%	0.0006%	0.0000%	0.0003%	0.0020%
password	0.0436%	0.0002%	0.0000%	0.0002%	0.0001%
123qwe	0.0277%	0.0005%	0.0105%	0.0001%	0.0013%
Sum (%)	3.0134%	**0.0393%**	**0.0177%**	**0.0093%**	**0.0101%**

†All the % are obtained by dividing the 8,129,445 Dodonew-tr accounts.

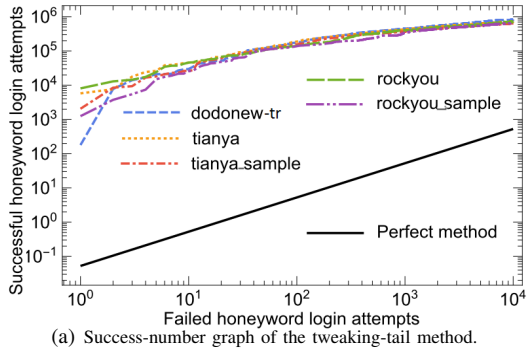
what will happen when the training set and test set come from more distinct distributions. To address this issue, we fix the attacking strategy to be "normalized top-PW with smoothing" (and set $\mathcal{T}_1$=1) and the test set to be Dodonew-ts, but with varying training sets: Dodonew-tr, Tianya and Rockyou. To understand the effects of the size of training sets, we also employ two additional training sets: Tianya_sample and Rockyou_sample, where Tianya_sample means a dataset randomly drawn from Tianya and its size equals Dodonew-tr, similarly for Rockyou_sample.

Fig. 4 shows the attacking results against the tweaking-tail method. In Fig. 4(a), when the honeyword login number allowed is small (e.g., <300), the attack trained on Tianya performs the best; As this number increases, the attack trained on Dodonew-tr performs the best. At 10^4 failed honeyword logins, the attack trained on Dodonew-tr can distinguish 820,703 real passwords, while the least effective attack, which is trained on Rockyou_sample, can still distinguish 642,668 real passwords, 1222 times higher than against the perfect method. Fig. 4(b) reveals that: (1) even trained and tested on two distant distributions (i.e., Tianya vs Dodonew-ts), the "norm top-PW" strategy can gain 26% of success rate with just *one* guess, 5 times higher than Juels and Rivest originally expected (i.e., 5%); and (2) the 5 training sets show a clear hierarchy in flatness: Dodonew-tr > Tianya > Tianya_sample > Rockyou > Rockyou_sample. We also experimented on the three other methods, and get similar results.

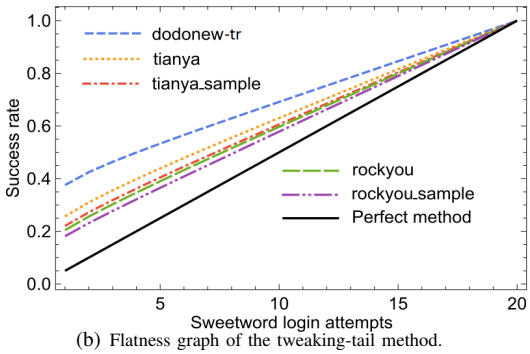
The high success-rates of English datasets (e.g., Rockyou) against the Chinese dataset Dodonew-ts (see Fig. 4(a) and 4(b)) are because: (1) Different kinds of "low-hanging fruits" can be revealed by the List model when trained on Dodonew-tr and Rockyou, respectively; (2) there are many internationally popular passwords(e.g., digital-sequences and keyboard-patterns) in both Dodonew-tr and Rockyou (see Fig. 3 of [30]). However, when measuring "flatness" (see Fig. 4(b)), the training set Rockyou is ineffective against Dodonew-ts.

In all, our results suggest that: (1) the closer the training set is to the test set, the more effective the attack will be; (2) the larger the training set, the more effective the attack will be; (3) under quite practical attacks, all the four methods examined provide far less security than expected.

Attacking password-model-based methods. Juels-Rivest's four primary methods are random-replacement based: the k-1 honeywords for a given user are generated by *randomly* replacing parts (or whole) of the real password. In *a passing comment*, Juels and Rivest [21] mention the possibility of using probabilistic password models (e.g., Weir et al.'s PCFG [32])



(a) Success-number graph of the tweaking-tail method.



(b) Flatness graph of the tweaking-tail method.

Fig. 4. An investigation of the effectiveness of different training sets: tested on Dodonew-ts, attacking with the "norm top-PW: smooth, 1t" strategy. Whenever a perfect method line is presented, the closer to the perfect line, the better the corresponding honeyword-method will be.

to build password model based honeyword methods. They explicitly left it as an open question (see Section 9 of [21]): "can the password models underlying cracking algorithms (e.g., [32]) be easily adapted for use?" Here we provide a negative answer to this question.

Here we take the well-known PCFG password model for example. We assume that, for a given user's password PW_i, the k-1 honeywords are produced according to the PCFG model (denoted by $P_{\mathsf{PCFG}(\mathcal{D})}(\cdot)$) trained on a specified dataset $\mathcal{D}$. Thus, the k-1 honeywords will be independent to PW_i.

In such methods, sweetwords with a higher probability generally will not be more likely to be the real password, because now the k-1 honeywords generated for PW_i are completely dependent on the password model (e.g., $P_{\mathsf{PCFG}(\mathcal{D})}(\cdot)$) but not PW_i: more popular events in the password model will be more likely selected as honeywords for PW_i.

As a result, the "Norm top-PW" strategy in Sec. III-B will not be effective. *We now propose a new attacking strategy, called "Norm PW-model" strategy*, by revising the "Norm top-PW" strategy. More specifically, all the settings in "Norm PW-model" strategy are the same with the "Norm top-PW" strategy, except for that (using the PCFG model for instance):

1) Add an additional Step 0 to the "Norm top-PW" attack strategy: the known password distribution $\mathcal{D}$ (e.g., a leaked password list) is first applied to the PCFG model (we use the laplace smoothing, see [24]), and this results in a password distribution $\mathsf{PCFG}(\mathcal{D})$.
2) Revise the Step 3 of "Norm top-PW" attack strategy to be: $\forall \mathrm{sw}_{i,j} \in \mathrm{SW}_i$, set

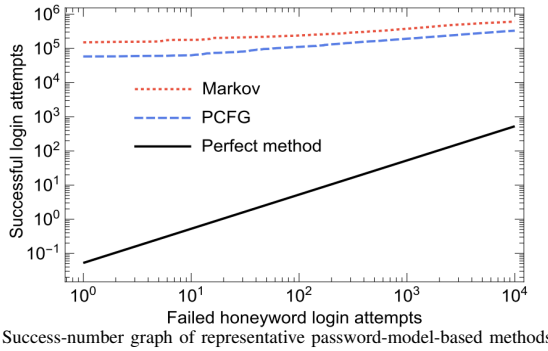
$$\Pr(\mathrm{sw}_{i,j}) = \frac{\frac{\Pr_{\mathcal{D}}(\mathrm{sw}_{i,j})}{\Pr_{\mathsf{PCFG}(\mathcal{D})}(\mathrm{sw}_{i,j})}}{\sum_{t=1}^{k} \frac{\Pr_{\mathcal{D}}(\mathrm{sw}_{i,t})}{\Pr_{\mathsf{PCFG}(\mathcal{D})}(\mathrm{sw}_{i,t})}}.$$

TABLE VII. SUCCESS NUMBER INFO AND ϵ-FLAT INFO ABOUT JUELS-RIVEST'S 4 METHODS (UNDER THE "NORM TOP-PW" STRATEGY) AND 2 PASSWORD-MODEL-BASED METHODS (UNDER THE "NORM PW-MODEL" STRATEGY): TRAINED ON DODONEW-TR, TESTED ON DODONEW-TS.

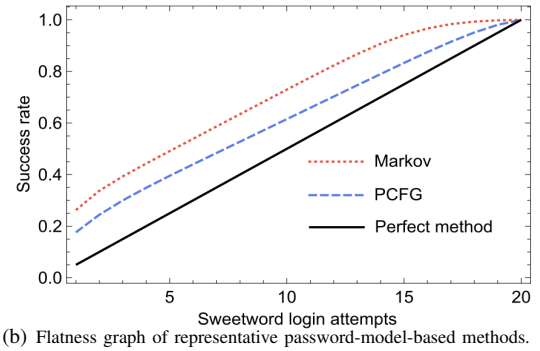
	Tweak tail	Model-syntax	Hybrid	Simple model	PCFG-based method	Markov-based method	Perfect method (base line)
Success number info	10.10%	9.06%	10.46%	8.75%	4.06%	7.52%	0.006%
ϵ-flat info	0.3755	0.4666	0.4063	0.3345	0.1761	0.2624	0.0500

For the Markov-model-based honeyword method, we can similarly define the "Norm PW-model" attack strategy.

As shown in Fig. 5(a), if $\mathcal{A}$ exploits our "Norm PW-model" strategy (trained on Dodonew-tr, tested on Dodonew-ts, and $\mathcal{T}_2 = 10^4$), honeyword-generation methods directly adapted from state-of-the-art password models are still vulnerable. More specifically, before $\mathcal{A}$ makes the first failed login attempt, she can successfully login 58,122 (0.71%) accounts against the PCFG-based method, and 150,459 (1.84%) accounts against the Markov-based method; before $\mathcal{A}$ makes the 10^4th wrong login attempts, she can successfully login 329,957 (4.06%) accounts and 611,388 (7.52%) accounts, respectively. Fig. 5(b) reveals that both methods are 0.1761^+-flat.



(a) Success-number graph of representative password-model-based methods.



(b) Flatness graph of representative password-model-based methods.

Fig. 5. An investigation of the effectiveness of two representative password-model-based honeyword-generation methods that are mentioned in [21]: trained on Dodonew-tr, tested on Dodonew-ts, attacking with the 'Norm PW-model" attack strategy (see Sec. III-E). Both methods are vulnerable.

As summarized in Table VII, the PCFG-based method performs the best among the six methods in terms of both the success number and ϵ-flatness metrics. Still, in terms of the success-number metric, it is 628=(329957/($\mathcal{T}_2$/(k-1))) times weaker than the optimal method; it is 0.1761-flat, three times weaker than the expected 0.05. It is likely that under more effective attacking strategies, the PCFG-based method will be even weaker. Naturally, an interesting question arises: is our "Norm PW-model" attacking strategy optimal for password-model based honeyword methods? Or, what's the optimal attacking strategy for password-model based methods? We left both questions as future research.

Summary. Our real-data grounded attacks show that the four methods examined all fall short of their expected security: under Juels-Rivest's assumed adversary model (i.e., a type-$\mathcal{A}_1$ attacker), the real passwords can be distinguished with a success rate of 29.29%∼32.62% by using our basic trawling-guessing attack, but not the claimed 5%, with just *one* guess when each user account is associated with 19 honeywords. We also provided a negative answer to the question of "can the password models underlying cracking algorithms (e.g., [32]) be easily adapted for use?" as left in [21].

IV. ADVANCED TRAWLING GUESSING ATTACKS

In the above, we have evaluated Juels-Rivest's four primary methods under two attack strategies by using the List password model (which is the most basic/simple password model). One defect of the List model is that, it is not good at eliminating "sweetword ties" (which are of similarly low probabilities) in uncommon sweetwords. In this Section, we address this issue by exploiting two sophisticated password models (PCFG and Markov) and smoothing techniques. Particularly, both models follow the Zipf's law (see Fig. 13(k) of [30]), and thus are good at assigning monotonically decreasing probabilities to strings/tokens to eliminate "sweetword ties".

A. Our no-training-set password model

Before presenting our advanced-model-based experiments, we propose a special password model for honeywords. It exploits the fact: if honeywords are uniformly distributed (i.e., $\Pr_{\mathrm{HW}}(\cdot|x)=\frac{1}{|T(x)|}$), then we can get the distribution of passwords from the distribution of sweetwords, where $T(x)$ denotes the sweetword space obtainable from the sweetword x and $\Pr_{\mathrm{HW}}(x)$ denotes the probability that x is chosen as the honeyword. We similarly define $\Pr_{\mathrm{PW}}(x)$ and $\Pr_{\mathrm{SW}}(x)$. Let $\widehat{\Pr}_{\mathrm{PW}}(x)$ denote an instantiation of $\Pr_{\mathrm{PW}}(x)$.

The sweetword list consists of $k-1$ honeywords and one password. Thus, the sweetword distribution $\Pr_{\mathrm{SW}}(\cdot)$ can be determined by the password distribution $\Pr_{\mathrm{PW}}(\cdot)$ and the honeyword distribution $\Pr_{\mathrm{HW}}(\cdot|\cdot)$ by using the equation:

$$\Pr_{\mathrm{SW}}(x) = \frac{1}{k}\Pr_{\mathrm{PW}}(x) + \frac{k-1}{k}\sum_y \Pr_{\mathrm{PW}}(y)\Pr_{\mathrm{HW}}(x|y).$$

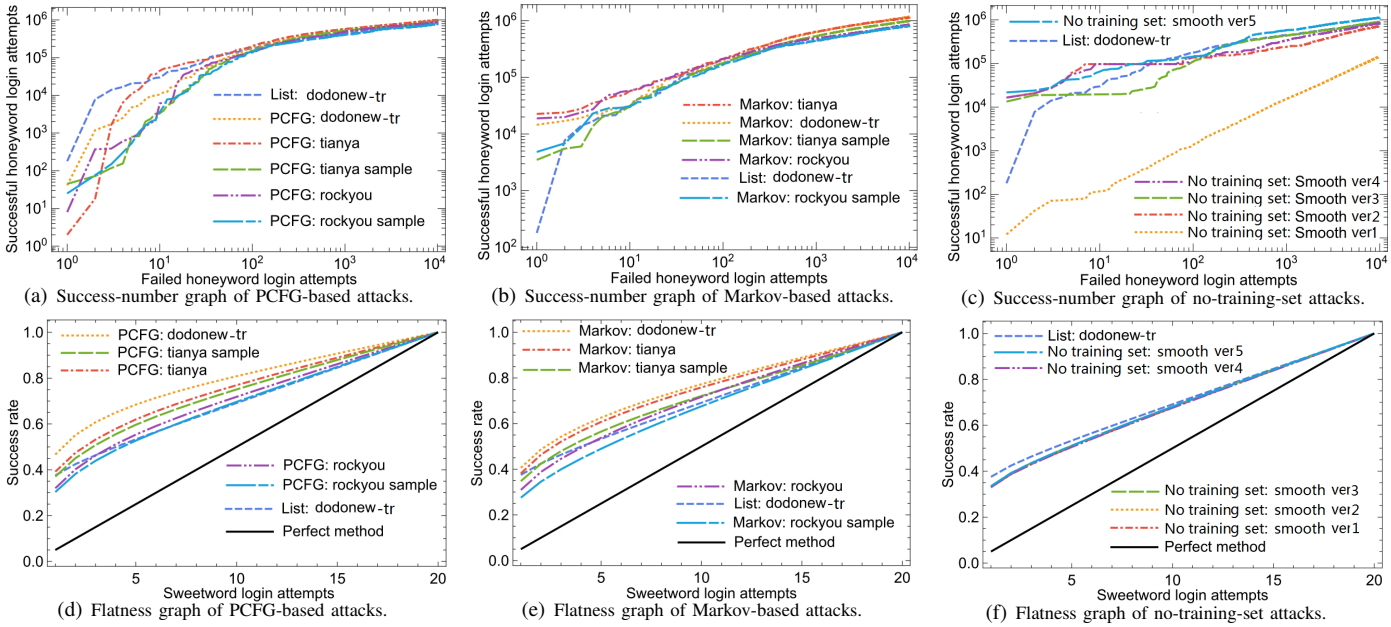
Then, we have

$$\begin{aligned}\Pr_{\mathrm{PW}}(x) =& k\Pr_{\mathrm{SW}}(x) - (k-1)\sum_y \Pr_{\mathrm{PW}}(y)\Pr_{\mathrm{HW}}(x|y)\\ =& k\Pr_{\mathrm{SW}}(x) - (k-1)\Pr_{\mathrm{PW}}(T(x))\frac{1}{|T(x)|}\\ =& k\Pr_{\mathrm{SW}}(x) - (k-1)\Pr_{\mathrm{SW}}(T(x))\frac{1}{|T(x)|}.\end{aligned}$$

Since the probability of a sweetword, denoted by $\Pr_{\mathrm{SW}}(x)$, can generally be approximated by its empirical probability $\frac{\mathrm{count}_{\mathrm{SW}}(x)}{\mathrm{count}_{\mathrm{SW}}}$, we get an *estimation* of $\Pr_{\mathrm{PW}}(x)$:

$$\widehat{\Pr}_{\mathrm{PW}}(x) = k\cdot\frac{\mathrm{count}_{\mathrm{SW}}(x)}{\mathrm{count}_{\mathrm{SW}}} - (k-1)\frac{\mathrm{count}_{\mathrm{SW}}(T(x))}{\mathrm{count}_{\mathrm{SW}}}\cdot\frac{1}{|T(x)|}.$$

As there are some sweetwords with $\widehat{\Pr}_{\mathrm{PW}}(\cdot)<0$, smoothing is needed. We tested five smoothing techniques for $\widehat{\Pr}_{\mathrm{PW}}(x)$ in Figs. 6(c) and 6(f), and results show ver5 performs the best.



(a) Success-number graph of PCFG-based attacks. (b) Success-number graph of Markov-based attacks. (c) Success-number graph of no-training-set attacks.

(d) Flatness graph of PCFG-based attacks. (e) Flatness graph of Markov-based attacks. (f) Flatness graph of no-training-set attacks.

Fig. 6. Experiment results for attacking the tweaking-tail method in [21] by using three sophisticated password models and five different training sets, with Dodonew-ts as the test set. The List password model trained on Dodonew-tr is used as the baseline.

$$\text{Ver1: } \widehat{\Pr}_{\text{PW}withSmooth1} = \max\{0, \widehat{\Pr}_{\text{PW}}(x)\};$$

$$\text{Ver2: } \widehat{\Pr}_{\text{PW}withSmooth2} = \max\{\tfrac{1}{\text{count}_{\text{SW}}}, \widehat{\Pr}_{\text{PW}}(x)\};$$

$$\text{Ver3: } \widehat{\Pr}_{\text{PW}withSmooth3} = \max\{\tfrac{\text{count}_{\text{SW}}(x)}{\text{count}_{\text{SW}}}, \widehat{\Pr}_{\text{PW}}(x)\};$$

$$\text{Ver4: } \widehat{\Pr}_{\text{PW}withSmooth4} = \max\{\tfrac{1}{\text{count}_{\text{PW}}}, \widehat{\Pr}_{\text{PW}}(x)\};$$

$$\text{Ver5: } \widehat{\Pr}_{\text{PW}withSmooth5} = \tfrac{\text{count}_{\text{SW}}(x)}{\text{count}_{\text{SW}}}.$$

B. Sophisticated password models

In the basic attacks in Sec. III, the probability of sweetwords comes from the List model—a known (leaked) password distribution $P_{\mathcal{D}}(\cdot)$. However, every leaked password distribution is of very limited space compared to the total password space. For instance, the currently known largest one is the 3 billion Yahoo leak [18], which is still far smaller than the space (about 10^{32}) of passwords that comply with a typical policy: consists of three kinds of characters and with $8 \leq \text{len} \leq 16$. Thus, one may conjecture that, under more sophisticated probabilistic password models (e.g., PCFG [32] and Markov [24]), the four methods in [21] will be even weaker. Here we follow the recommendations in [24], and use Laplace smoothing for the PCFG model and Backoff smoothing for the Markov model.

We now establish this conjecture. We design a series of attacks (see Fig. 6) against the tweaking-tail method to explore the effectiveness of various password-cracking models, including PCFG [32], Markov [24] and our above no-training-set ones. In our Markov-based attacks, all the settings are the same with the "Norm top-PW: smooth, 1t" in Fig. 3(a), except for that: *the known password distribution $\mathcal{D}$ (e.g., a leaked password list) is first applied to the Markov cracking model (we use the backoff approach, see [24]), and this results in a password distribution* $\mathsf{Markov}(\mathcal{D})$*, then $P_{\mathsf{Markov}(\mathcal{D})}(\cdot)$ is used instead of $P_{\mathcal{D}}(\cdot)$ to assign probabilities*. We similarly denote the PCFG-based password model to be $P_{\mathsf{PCFG}(\mathcal{D})}(\cdot)$ and the no-training-set model to be $P_{\mathcal{F}}(\cdot)$. See the title of Fig. 6 for how these models were trained and tested.

TABLE VIII. SUCCESS-NUMBER INFO (%) ABOUT THE TWEAKING-TAIL METHOD BY USING FOUR ADVANCED PASSWORD CRACKING MODELS AND FIVE DIFFERENT TRAINING SETS, UNDER THE (TYPE-$\mathcal{A}_1$) "NORMALIZED TOP-PW ATTACK: 1T" ATTACK, TEST SET=DODONEW-TS, AND $\mathcal{T}_2 = 10^4$.†

	List	PCFG	Markov	No-training-set
Dodonew-tr	10.10%	12.45%	**14.48%**	13.80% (using ver5)
Tianya	8.94%	12.22%	**14.01%**	11.06% (using ver3)
Tianya_sample	7.91%	11.15%	**12.23%**	10.43% (using ver4)
Rockyou	8.86%	**10.73%**	10.51%	8.87% (using ver2)
Rockyou_sample	8.05%	9.59%	**9.94%**	1.79% (using ver1)

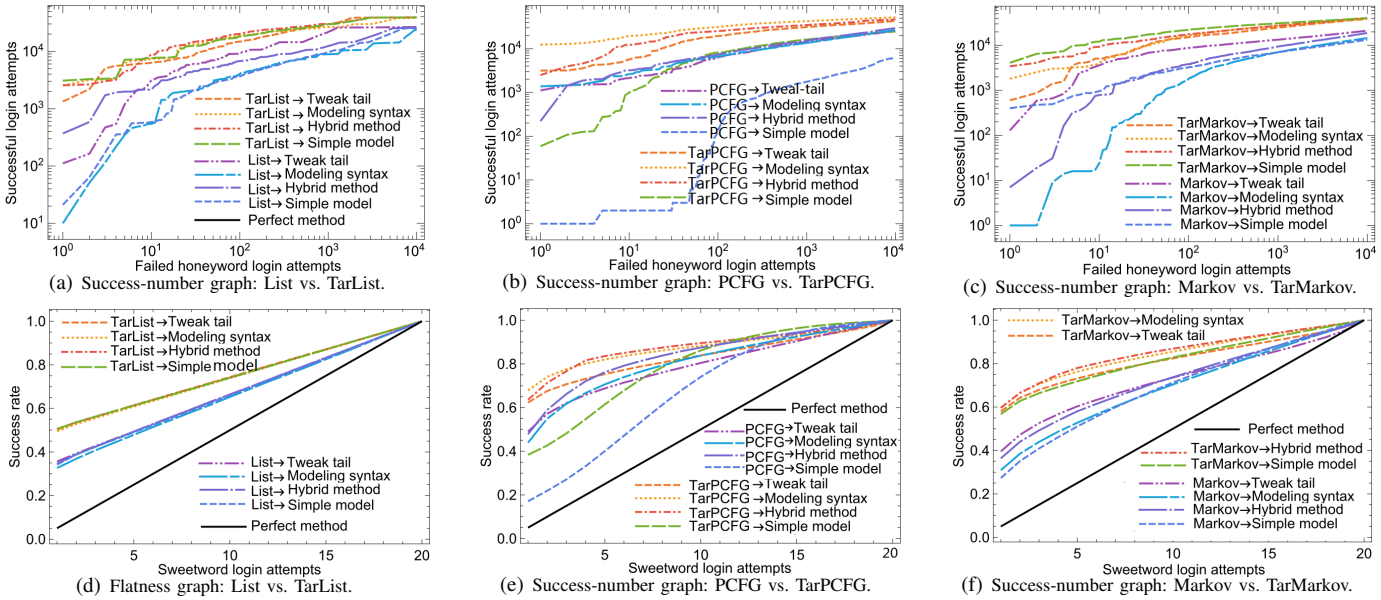
†A value in bold means the corresponding password model performs best. All the % are obtained by dividing the 8,129,445 Dodonew-tr accounts.

TABLE IX. ϵ-FLAT INFO ABOUT THE TWEAKING-TAIL METHOD. ALL THE EXPERIMENTAL SETTING ARE THE SAME WITH TABLE VIII.

	List	PCFG	Markov	No-training-set
Dodonew-tr	0.3755	**0.4666**	0.4063	0.3345 (using ver5)
Tianya	0.2565	**0.3917**	0.3817	0.2201 (using ver3)
Tianya sample	0.2204	**0.3704**	0.3480	0.2345 (using ver4)
Rockyou	0.2044	**0.3194**	0.3094	0.2030 (using ver2)
Rockyou sample	0.1813	**0.3029**	0.2753	0.1743 (using ver1)

As shown in Fig. 6, and summarized in Table VIII and Table IX: (1) in general, the PCFG-based model $P_{\mathsf{PCFG}(\mathcal{D})}(\cdot)$ and the Markov-based model $P_{\mathsf{Markov}(\mathcal{D})}(\cdot)$ perform better than the List model $P_{\mathcal{D}}(\cdot)$, when given the same training set and test set; (2) the tweaking-tail method is 0.47^{+}-flat (see Fig. 6(d)), 9 times weaker than expected in [21]; and (3) surprisingly, our no-training-set model $P_{\mathcal{F}}(\cdot)$ (with smoothing version 5) well approaches the List model $P_{\mathcal{D}}(\cdot)$, indicating that $\mathcal{A}$ can even perform effective attacks by just using the sweetword file F (and needs no external training sets).

Summary. Results show that Juels-Rivest's four methods are much weaker under our more advanced trawling-guessing attackers who can exploit various state-of-the-art password models like PCFG [32] and Markov [24]. More specifically, under the adversary model (i.e., a type-$\mathcal{A}_1$ attacker) assumed by Juels and Rivest, the real passwords can be distinguished with a success rate of 34.21%∼49.02% (see the left category of Table X), but not the claimed 5%, with just *one* guess when each user account is associated with 19 honeywords.



(a) Success-number graph: List vs. TarList. (b) Success-number graph: PCFG vs. TarPCFG. (c) Success-number graph: Markov vs. TarMarkov.

(d) Flatness graph: List vs. TarList. (e) Success-number graph: PCFG vs. TarPCFG. (f) Success-number graph: Markov vs. TarMarkov.

Fig. 7. Evaluating Juels-Rivest's methods under type-$\mathcal{A}_2$ attackers, trained on PII-Dodonew-tr, tested on PII-Dodonew-ts. Type-$\mathcal{A}_2$ attackers perform much better than type-$\mathcal{A}_1$ ones: a clear security hierarchy appears. Sub-figure 7(f) reveals that, under a Type-$\mathcal{A}_2$ attacker, every method in [21] is 0.568^{+}-flat.

V. Targeted Guessing Attacks

In the above, we have evaluated Juels-Rivest's four main honeyword methods [21] under their assumed adversary model (i.e., a type-$\mathcal{A}_1$ attacker). As users love to build their (real) passwords with PII and such PII is becoming increasingly easy to be learned by $\mathcal{A}$ through various means (see Sec. II-A), a natural question arises: *What's the security of these methods when $\mathcal{A}$ is further equipped with the knowledge of user PII (i.e., under a type-$\mathcal{A}_2$ attacker)?* We now answer this question.

We employ three targeted password-cracking models: TarPCFG [30], TarList and TarMarkov. Here TarPCFG is just the TarGuess-I in [30], while TarList and TarMarkov are *PII-enriched* Markov model and List model, respectively. The TarMarkov model can be converted from the Markov model by applying the type-based PII segment matching method as proposed in [30]. To accomplish the conversion, we only need to incorporate the various type-based PII tags $\{\mathsf{N}_1, \cdots, \mathsf{N}_7; \mathsf{B}_1, \cdots, \mathsf{B}_{10}; \mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3; \cdots\}$ defined in [30] into the alphabet Σ (e.g., $\Sigma = \{95$ printable ASCII characters$\}$ in [24]) of the Markov n-gram model. Then, all operations for these type-based PII tags are the same with the original characters in Σ. Note that, the PII tag N stands for name usages, B for birthday, A for account name, and so on; The subscript of a PII usage means a specific *type* but not *length*, e.g., N_1 stands for full name "`john smith`" and N_2 for the abbrev. of full name: `js`←"`john smith`". Similarly, we can convert the List model into the TarList model.

In our TarPCFG-based attacks, all the settings are the same with the "Norm top-PW: smooth, 1t" in Fig. 3(a), except for that: the known password distribution $\mathcal{D}$ (e.g., a leaked password list) is first used as the training set to the TarPCFG password model [30], and this results in a password distribution $P_{\mathsf{TarPCFG}}(\mathcal{D})$. Then, $P_{\mathsf{TarPCFG}(\mathcal{D})}(\cdot)$ is used instead of $P_{\mathcal{D}}(\cdot)$ to assign sweetword probabilities. We similarly denote the TarMarkov model to be $P_{\mathsf{TarMarkov}(\mathcal{D})}(\cdot)$ and the TarList model to be $P_{\mathsf{TarList}}(\cdot)$. Note that, each password model is trained on PII-Dodonew-tr and tested on PII-Dodonew-ts.

Fig. 7 compares the effectiveness of type-$\mathcal{A}_1$ and type-$\mathcal{A}_2$ attackers against Juels-Rivest's methods. Within each subfigure, it is evident that type-$\mathcal{A}_2$ attackers are substantially more effective than type-$\mathcal{A}_1$ ones. More specifically, $\mathcal{A}$ can guess 57.6%∼96.2% more real passwords when $\mathcal{T}_2$=10^4, and achieve 27.2%∼65.9% more success rates (see Table X) in terms of the ϵ-flatness metric. Alarmingly, against every method in [21] and based on some targeted-guessing password model, PII-enriched $\mathcal{A}$ now can reach 56.8%∼67.9% of success rates in telling apart the real password from 19 honeywords by making just *one* online guess (see the point (x=1, y=0.5) in Fig. 7(f)), while the claimed (desired) security is about 5%. In all, honeyword methods that do not consider user PII cannot withstand type-$\mathcal{A}_2$ attackers that exploit the victim's PII.

TABLE X. ϵ-Flatness info under type-$\mathcal{A}_1$ and $\mathcal{A}_2$ attackers.[†]

Method	Type-$\mathcal{A}_1$ Attacker			Type-$\mathcal{A}_2$ Attacker			Increase
	List	PCFG	Markov	TarList	TarPCFG	TarMarkov	
Tweak-tail	0.356	**0.490**	0.395	0.502	**0.623**	0.580	**127.2%**
Model-syntax	0.326	**0.439**	0.309	0.495	**0.679**	0.597	**154.4%**
Hybrid	0.346	**0.478**	0.364	0.506	**0.636**	0.596	**133.0%**
Simple model	**0.342**	0.171	0.273	0.505	0.384	**0.568**	**165.9%**

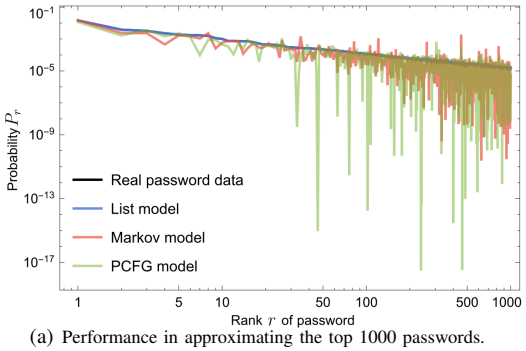
[†]A value in bold means that the corresponding password model perform the best among its attacker category. The "Increase" column is computed by dividing the best-case value under a type-$\mathcal{A}_2$ attacker with that of a type-$\mathcal{A}_1$ attacker, e.g., 127.2%=0.623/0.490.

Summary. We have evaluated Juels-Rivest's four methods under a type-$\mathcal{A}_2$ attacker, a powerful yet realistic attacker, by employing three targeted password-cracking models. Our results show that, these four methods all perform poorly: the real passwords can be distinguished with a success rate of 56.8%∼67.9% with just *one* online guess when each user account is associated with 19 honeywords, under the assumption that $\mathcal{A}$ knows some of the victim's common PII.

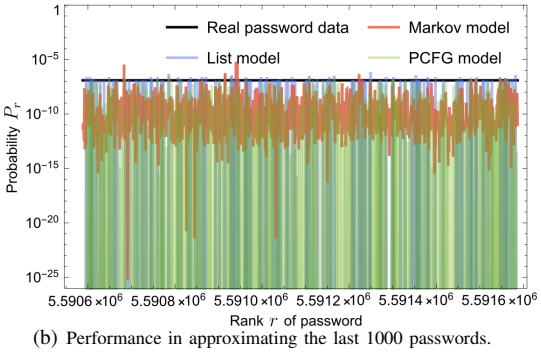
VI. Potential Countermeasures

Only when we know how to attack, can we know how to protect. Our above attacking results highlight three key contributions to harden the process of generating honeywords.

An impossibility result. As said in Sec. III-A, all Juels-Rivest's four methods essentially belong to the random re-

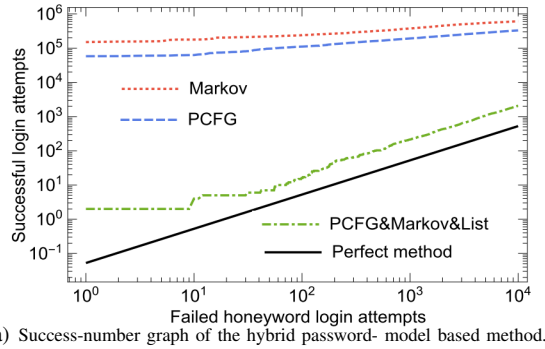


(a) Performance in approximating the top 1000 passwords.

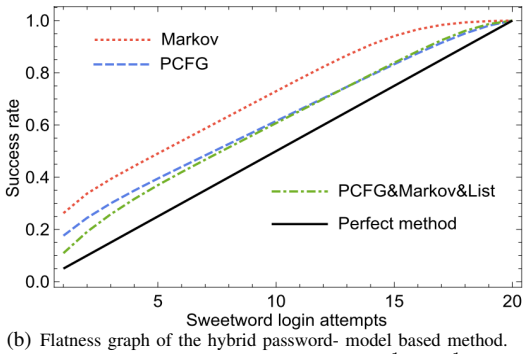


(b) Performance in approximating the last 1000 passwords.

Fig. 8. An investigation of the pros and cons of different password models, trained on 8.129M Dodonew-tr and tested on 8.129M Dodonew-ts.



(a) Success-number graph of the hybrid password- model based method.



(b) Flatness graph of the hybrid password- model based method.

Fig. 10. Investigating the hybrid password model $\frac{1}{3}$List+$\frac{1}{3}$Markov+ $\frac{1}{3}$PCFG, trained on 8.129M Dodonew-tr and tested on 8.129M Dodonew-ts.

placement based honeyword-generation approach: each honeyword is generated by randomly replacing parts (or whole) of the real password PW_i. Let $T(\mathrm{PW}_i)$ denote the sweetword space resulted from PW_i, and $\mathcal{D}$ denote the password space. We have $T(\mathrm{PW}_i) \subseteq \mathcal{D}$. For instance, the tweak-tail method implies $T(123456) = \{123t_1t_2t_3 | t_1, t_2, t_3 \in \{0, 1, \ldots, 9\}\}$. Under the random-replacement approach, each of the $k-1$ *honeywords* in the sweetword list SW_i can be seen as *randomly* drawn from $T(\mathrm{PW}_i)$, and thus they each have the *same* probability (i.e., $\frac{1}{|T(\mathrm{PW}_i)|}$) to be chosen as a honeyword. If the real password PW_i is also *randomly* drawn from $T(\mathrm{PW}_i)$, then the random-replacement approach will be perfect.

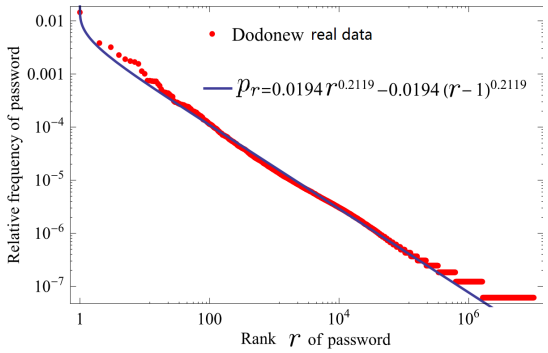


Fig. 9. Fitting Dodonew passwords by using the CDF-Zipf model [28]. The probability of popular passwords decrease monotonically.

However, users generally do *not* randomly choose their real passwords from $T(\mathrm{PW}_i)$, but follow the highly skewed Zipf's law (see [28]) in $\mathcal{D}$: $p_r = C \cdot r^s - C \cdot (r-1)^s \approx C \cdot s \cdot r^{s-1}$, where p_r denotes the probability of the rth popular password in $\mathcal{D}$, and $s \in [0.15, 0.30]$ and $C \in [0.001, 0.1]$ are constants depending on $\mathcal{D}$. It is easy to see that $\forall x_1, x_2 \in T(\mathrm{PW}_i)$, if $x_1, x_2 \in \mathrm{SW}_i$ and $\Pr(x_1) > \Pr(x_2)$, then x_1 is more likely to be the real password than x_2. Both our "Top-PW" and "Norm Top-PW" attacking strategies tend to rank x_1 higher than x_2, and thus will successfully pick x_1 out earlier.

To resist such attacks, a honeyword method should ensure that the elements in $T(\mathrm{PW}_i)$ satisfy: $\forall x_1, x_2 \in T(\mathrm{PW}_i)$, $\Pr(x_1) = \Pr(x_2)$. Unfortunately, the Zipf's law reveals that p_r is a monotonically decreasing function with r, which is particularly true for popular passwords (see Fig. 9 for a concrete example). Thus, when given the real password PW_i, it is *inherently difficult* to produce $|T(\mathrm{PW}_i)|$ sweetwords with the same probability of $\Pr(\mathrm{PW}_i)$. This indicates that random-replacement methods (including Juels-Rivest's) are impossible to achieve expected security and cannot be easily remedied.

A counter-intuitive insight. The above impossibility result suggests us to adopt the password model based approach. However, in Sec. III-E we have demonstrated that, as opposed to common belief (as hold in [21]), existing probabilistic password models cannot be readily used as honeyword generation methods. Now we further investigate the underlying reason.

The goodness of a password model depends on how it approximates user password behaviors. Fig. 8 shows that each password model has its own weaknesses: (1) the List model is good at approximating the probability of popular passwords, yet worst at least popular passwords; (2) the PCFG model is pretty good at approximating the probability of least-popular passwords, yet worst at top popular passwords; and (3) The Markov model lies between them.

A possible solution. The above findings naturally lead us to propose a hybrid approach: combine varied password models to avoid individual defect. The key issue lies in how to effectively combine them. We conduct a preliminary experiment by using the simplest (i.e., linear) combination: $\frac{1}{3}$List+$\frac{1}{3}$Markov+ $\frac{1}{3}$PCFG. Though simple in its nature, the hybrid password model greatly improves security: only 1 time more real passwords can be identified than the perfect method

when $\mathcal{T}_2$=10^4 (i.e., 1113 vs 526, see Fig. 10(a)) and being 0.11-flat (see Fig. 10(b)). This suggests a promising direction for the research community. Also note that, if there exist more effective attacking strategies, the hybrid password model might be less secure than it currently appears, and thus developing honeyword attacking theories is important future research.

VII. Conclusion

We have for the first time empirically evaluated the four primary honeyword-generation methods proposed by Juels and Rivest at CCS'13, by using both trawling guessing attackers with no user PII and targeted guessing attackers with some common user PII. We showed that their methods all fail to meet the expected security level by a large margin. We also provided a negative answer to the open question of "can the password models underlying cracking algorithms (e.g., [32]) be easily adapted for use?" left by Juels and Rivest.

We consider the value of our work mainly two-folds. First, our work is the first to give comprehensive, convincing quantitative evidence of how vulnerable the state-of-the-art honeyword-generation approaches are, highlighting the deceptively simple nature of honeyword research—it is not easy, but a great challenge to automatically generate honeywords that are hard to differentiate from real passwords. Second, as the limitations of the four honeyword methods examined cannot be readily addressed, our work calls for novel and creative efforts to contribute to principled honeyword-generation approaches that are significantly better than the current heuristic approaches. Considering the prevalence and catastrophic impacts of password-file leakage, we believe that achieving active and timely compromise detection is of broad interest, and our work constitutes an important step forward in this direction and will trigger interest for new honeyword research.

Acknowledgment

The authors are grateful to the shepherd, Prof. Adam Aviv, of our paper. We also thank the reviewers for their constructive comments. Ping Wang is the corresponding author. This research was supported by the National Natural Science Foundation of China under Grants Nos. 61472016 and 61472083, by the National Key Research and Development Plan under Grants Nos. 2016YFB0800603 and 2017YFB1200700, and by the Wallenberg Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] S. Achappell, *Turkey: personal data of 50 million citizens leaked online*, April 2016, http://www.euronews.com/2016/04/06/turkey-personal-data-of-50-million-citizens-leaked-online-hackers-claim.

[2] 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.

[3] J. Blocki, S. Komanduri, L. Cranor, and A. Datta, "Spaced repetition and mnemonics enable recall of multiple strong passwords," in *Proc. NDSS 2015*, pp. 1–15.

[4] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, "Kamouflage: Loss-resistant password management," in *Proc. ESORICS 2010*, pp. 286–302.

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

[6] 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.

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

[8] X. Carnavalet and M. Mannan, "From very weak to very strong: Analyzing password-strength meters," in *Proc. NDSS 2014*, pp. 1–16.

[9] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive password-strength meters from markov models," in *Proc. NDSS 2012*, pp. 1–15.

[10] N. Chakraborty and S. Mondal, "On designing a modified-UI based honeyword generation approach for overcoming the existing limitations," *Comput. Secur.*, vol. 66, pp. 155–168, 2017.

[11] M. Dürmuth, D. Freeman, and B. Biggio, "Who are you? A statistical approach to measuring user authenticity," in *NDSS 2016*, pp. 1–15.

[12] M. Dürmuth and T. Kranz, "On password guessing with GPUs and FPGAs," in *Proc. Password 2014*, pp. 19–38.

[13] V. Enterprise, *2016 Data Breach Investigations Report*, May 2017, http://www.verizonenterprise.com/resources/reports/rp_dbir-2016-executive-summary_xg_en.pdf.

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

[15] J. Goldberg, "Bcrypt is great, but is password cracking infeasible?" Mar. 2015, http://t.cn/RGYUtJO.

[16] J. Goldman, *Chinese Hackers Publish 20 Million Hotel Reservations*, Dec. 2013, http://bit.ly/2aVKyBw.

[17] M. Golla, B. Beuscher, and M. Dürmuth, "On the security of cracking-resistant password vaults," in *Proc. ACM CCS 2016*, pp. 1230–1241.

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

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

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

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

[22] *How Many Times Has Your Personal Information Been Exposed to Hackers?*, Sep. 2016, http://nyti.ms/1SdFv0s.

[23] R. W. Lai, C. Egger, D. Schröder, and S. S. Chow, "Pheonix: Rebirth of a cryptographic password-hardening service," in *Proc. Usenix SEC 2017*, pp. 899–916.

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

[25] *Equifax Data Breach Impacts 143 Million Americans*, Sep. 2017, https://www.forbes.com/sites/leemathews/2017/09/07/equifax-data-breach-impacts-143-million-americans.

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

[27] S. Ragan, *Weebly data breach affects 43 million customers*, Oct. 2016, http://bit.ly/2kP4EA2.

[28] 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.

[29] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," IEEE Trans. Depend. Secur. Comput, 2016, Doi:10.1109/TDSC.2016.2605087.

[30] 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.

[31] C. Weir, *Cracking the MySpace List–First Impressions*, July 2016, http://reusablesec.blogspot.com/2016/07/cracking-myspace-list-first-impressions.html.

[32] 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.

[33] B. B. Zhu, J. Yan, D. Wei, and M. Yang, "Security analyses of click-based graphical passwords via image point memorability," in *Proc. ACM CCS 2014*, pp. 1217–1231.