On the Insecurity of Internally Sampled Honeyword Schemes

Ding Wang, Tingwei Fan, Fei Duan, and Zhenduo Hou

***Abstract*—Honeywords are plausible-looking decoy passwords associated with each user's real password to timely detect password leakage. The more indistinguishable the honeywords are, the more secure a honeyword scheme is. However, honeyword schemes that externally generate honeywords can only approximate, but not equate, the distribution of user-chosen passwords, so they are unlikely to achieve the ideal indistinguishability. To address this issue, internally sampled honeyword schemes that sample honeywords from other users' passwords have been proposed.**

In this work, we first reveal two critical security and two critical usability flaws in existing internally sampled honeyword schemes, i.e., Honeyindex (TDSC'16) and Superword (COSE'21). We then formalize a *generic* framework for sound internally sampled honeyword schemes, and propose variants for both Honeyindex and Superword. To principally evaluate the security of our framework, we propose Bayesian and intersection attack theories leading to attackers' optimal distinguishing strategies, and evaluate them under three major attacker models each with varied capabilities (e.g., using leaked datasets and users' personal information). Evaluation results show that, when 40 sweetwords are associated with each user (as recommended at IEEE S&P'22), with only one guess per account, the basic attacker's success rate can reach 3.82%∼4.12%, and she can identify 4.31%∼5.04% of all real passwords with 10^4 honeyword login attempts, breaking the ideal 2.50%(=1/40) security. Two more advanced attackers can identify 18.6%∼44.8% and 20.6%∼43.6% of all real passwords in 10^4 honeyword login attempts, respectively. When multiple password files are available, the intersection attack alone identifies 18.3%∼18.6% of real passwords. We also explore the impacts of denial-of-service attacks. In all, this work reveals the *inherent* insecurity of internally sampled honeyword schemes.

***Index Terms*—Honeyword; Internally sampled scheme; Honeyword distinguishing attacks; Bayesian attack; Intersection attack.**

I. Introduction

Passwords are the most prevalent user authentication method. Although they suffer from both security (e.g., guessing [37], [40], phishing [26], [30]) and usability (e.g., memorization [14]) issues, no alternatives (e.g., biometric [20], [25] and FIDO2 [1]) can simultaneously provide high availability, low deployment cost, and ease of change benefits. Therefore, both academia and industry have gradually realized that passwords will still exist in the foreseeable future [4], [5], [12].

This work was supported in part by the National Natural Science Foundation of China under Grant ~~62222208 and~~ 62572259, and in part by Tianjin Natural Science Foundation Project (Grant No. 25JCJQJC00230). The corresponding author is Zhenduo Hou.

Ding Wang, Tingwei Fan, Fei Duan, and Zhenduo Hou are with the College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China, and also with Key Laboratory of Data and Intelligent System Security (Ministry of Education), Nankai University, Tianjin 300350, China, and also with Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China. (Email: wangding@nankai.edu.cn; fantingwei@mail.nankai.edu.cn; duanf@mail.nankai.edu.cn; joehou13@nankai.edu.cn).

In a password-based authentication system, the authentication server stores the sensitive password file, offering high-profile targets for cyber attacks. Notably, well-known companies such as Yahoo [19], [27], [46], and LastPass [29], [31], [44] have suffered from multiple password file leakages. Worse still, such leakages are hard to detect, enabling attackers to have sufficient time to crack passwords and exploit them. For instance, millions of Yahoo passwords were leaked in 2012, 2013, and 2014, and remained undetected for 1∼4 years [19], [27], [46]. The LastPass password files were also leaked in 2011, 2015, and 2022, and remained undetected for over half a year [29], [31], [44]. Such leakages often lead to catastrophic consequences. For instance, LastPass users have lost cryptocurrency worth $4.4 million in a single day, and 1200 Bitcoins (valued at $32 million when the leakage happened) were stolen in total [28]. *All this underscores the necessity to timely detect password leakage and take responsive countermeasures.*

To achieve this goal, Juels and Rivest proposed the honeyword scheme [18]. In general, it mixes each user's real password with $k-1$ (e.g., k=40 [38]) decoy passwords (called honeywords) designed to be indistinguishable. Each user's real password and $k-1$ decoy passwords are collectively referred to as sweetwords. For each user, the index of the real password among sweetwords is stored as a *secret* on a trusted secure server called honeychecker. In this case, even if the attacker can successfully recover sweetwords offline (e.g., by using advanced hardware such as GPU and FPGA [8], [11] and software like Prob-Hashcat [16]), she still needs to distinguish the real password from honeywords (i.e., honeyword distinguishing attacks) by using online verifications to find if the chosen one is real. Logins with honeywords signal potential password file leakages, enabling security administrators to take responsive countermeasures (e.g., pushing breach alarms).

By design, the security of a honeyword scheme lies in the indistinguishability of honeywords from a user's real password, which is challenging to achieve [17], [35], [38]. Broadly, the technical routes of existing honeyword schemes can be categorized as externally generated and internally sampled ones. The first obtains honeywords by using external algorithms (e.g., PCFG [43] and Markov [24]) trained on external datasets, *and sweetwords of each user are independent.* The second obtains honeywords by sampling from other users' passwords *within* the same website, and has two major differences: (1) Sweetwords of different users are correlated, as a user's real password can be another user's honeyword; (2) Duplicate sweetwords can exist, due to users' usage of popular passwords and the nature of random sampling. *These differences impact the security and usability of internally sampled honeyword schemes.*

On the one hand, the externally generated route can be divided into random-replacement- and password-model-based honeyword generation techniques (HGTs). As revealed in [35], [38], random-replacement-based HGTs (i.e., creating honeywords by randomly replacing characters in externally generated honeyword candidates like `snurfle672`→`snurfle806`) cannot reach the designed security goal. In particular, at NDSS'18, Wang et al. [35] found that even for a basic attacker that only knows the publicly leaked password datasets, with just one guess, the success rate of identifying a user's real password can be 29.29%∼32.62%, not the designed 5% (i.e., $1/k$, when k=20) in [18]. For a more advanced targeted-guessing attacker who can leverage victims' personal information, the success rate can be as high as 56.81%∼67.98% [35].

For the password-model-based HGTs, honeywords are selected from externally generated password models (e.g., PCFG [43] and Markov [24]). At IEEE S&P'22, Wang et al. [38] first proposed an attack theory, and then HGTs consisted of hybrid password models, i.e., $\frac{1}{3}$(Tar)List+ $\frac{1}{3}$(Tar)PCFG+$\frac{1}{3}$(Tar)Markov (where "Tar" means the targeted model that utilizes users' personally identifiable information such as names and birthdates [36], [37]). Each model is selected with a probability of $1/3$, and each password (with duplicates) in a model is uniformly selected as a honeyword. However, such HGTs can only reduce attackers' one-guess success rate to 15%, which is still far from the designed 5% and the ideal $1/k$ goals (e.g., k=40 [38]). At NDSS'24, Wang and Reiter [41] proposed Bernoulli honeywords, where each unique (i.e., without duplicates) password in a model (e.g., TarGuess-I [37], see Fig.1-(a) of [41]) is selected as a honeyword with a fixed probability $p_h \approx 10^{-8}$ and can be analyzed similarly to Wang et al.'s naive Bayesian attack that infers sweetwords [38]. Due to the necessity of employing external password models, both Wang et al.'s (e.g., List, which directly samples honeywords from a leaked password dataset) [38] and Wang-Reiter's Bernoulli HGTs [41] belong to the externally generated route.

On the other hand, the internally sampled route obtains honeywords by directly sampling from other users' real passwords within the same website. Unlike externally generated schemes that independently generate honeywords for each user account, the server of an internally sampled scheme stores the indices of how his/her honeywords are sampled from others' real passwords. This means that a user's real password can be another user's honeyword, and vice versa. In this manner, internally sampled honeywords are believed to be distributed more closely to users' real passwords, thus achieving better indistinguishability [13], [15]. In particular, Honeyindex [13] and Superword [15] are claimed to achieve the ideal security, where an attacker cannot develop a more efficient attack strategy than random guessing. However, the security of internally sampled honeyword schemes has only been analyzed in ad-hoc manners, which calls for a generic formalization and a rigorous attack theory. To the best of our knowledge, we are the first to systematically deal with this key research question.

A. Motivations

First proposed by Juels and Rivest, honeyword generation techniques (HGTs) are decoy passwords used to detect password leakages [18]. Depending on how honeywords are generated, existing honeyword schemes can be categorized into externally generated and internally sampled ones. Existing studies (see [10], [18], [35], [38], [41]) mainly focus on the first one, but find that HGTs of this category are far from the designed security goal (e.g., attackers' one-guess success rates far exceed the designed 5% in [18]). This motivates the security administrators towards internally sampled honeyword schemes, where a user's honeywords are sampled from others' real passwords. Though it looks plausible that honeywords of this category are more similarly distributed to real passwords, to the best of our knowledge, existing schemes (e.g., Honeyindex [13] and Superword [15]) are of diverse forms, and their security analyses are mostly ad-hoc. We explain the details as follows.

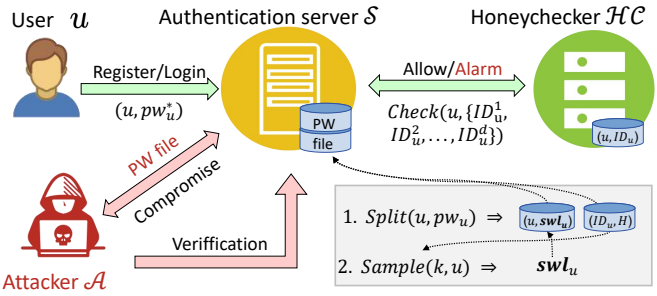


Fig. 1: An illustration of the internally sampled honeyword framework. After a user u registers with her password pw_u, the authentication server $\mathcal{S}$ splits (u, pw_u) into (u, ID_u) and (ID_u, H), where ID_u is her index and H is the salted-hash $H(pw_u||salt_u)$. Then, $\mathcal{S}$ stores (u, ID_u) on the honeychecker $\mathcal{HC}$, randomly samples $k-1$ indices from other users, and uses their passwords as u's honeywords. Upon logging in with pw_u^*, $\mathcal{S}$ sends the index set $\{ID_u^1, ID_u^2, \cdots, ID_u^d\}$ of matched sweetwords to the honeychecker $\mathcal{HC}$, and $\mathcal{HC}$ checks if $ID_u \in \{ID_u^1, ID_u^2, \cdots, ID_u^d\}$ holds. The attacker $\mathcal{A}$ who only compromises $\mathcal{S}$ cannot identify a user's real index/password.

Firstly, rigorous security analysis of internally sampled honeyword schemes is missing. For Honeyindex, Erguler [13] falsely treated honeywords of different users as independent, and claimed that Honeyindex can achieve the ideal $1/k$ security, i.e., the attacker $\mathcal{A}$ can only randomly choose one of the k sweetwords as the real password in distinguishing attacks. Later, Wang et al. [35] proposed the peeling-onion distinguishing attack that achieves a 100% success rate, but its effectiveness depends on $\mathcal{A}$'s capability of finding an onion (i.e., an isolated sweetword that only appears in one user's sweetword list). However, when the regular update mechanism (i.e., periodically resampling honeywords, as recommended in [13]) is enforced, onions can be hard to find [38] (see Sec. IV-B). Moreover, due to duplicate sweetwords, false authentication denials may occur when the user's password is equal to one or more sampled honeywords. Unfortunately, this critical design flaw has not been investigated in existing studies [13], [35], [38].

For Superword, Guo et al. [15] ignored serious design flaws. To begin with, Guo et al. failed to identify the drawbacks entailed by storing user-specific salts on the honeychecker: Superword can neither detect individual password leakage, nor maintain high detection rates (i.e., resisting distinguishing attacks) and low false-alarm rates (i.e., resisting DoS attacks). Additionally, the risk posed by the honeychecker failure remains uninvestigated. Besides, Guo et al. also ignored the impacts of

skewed (e.g., CDF-Zipf [34]) password distributions, and falsely claimed that Superword can achieve the ideal $1/n$ (where n is the number of user accounts) security [15] (see Sec. III-B). These flaws call for a generic framework for internally sampled schemes that can incorporate sound Superword variants.

Secondly, as shown in [9], [42], an attacker may breach the authentication server multiple times and obtain different snapshots of the password file, but the impacts have not been rigorously considered. For instance, in Honeyindex [13], regular updates require the server to resample honeywords for each user after a new batch of users registers, so a sweetword list can differ before/after resamplings. In such a case, an attacker can compare each user's sweetword lists in different password files, and devise attack strategies to facilitate distinguishing attacks (see Sec. IV-D). This calls for rigorous attack theories as well as extensive experiments to evaluate the real-world security of internally sampled honeyword schemes.

B. Contributions

- **A generic framework for designing sound internally sampled honeyword schemes.** We reveal two critical security and two critical usability flaws in existing Honeyindex [13] and Superword [15] internally sampled honeyword schemes, and show their impacts. We then propose a generic framework for designing sound internally sampled honeyword schemes and variants for both Honeyindex (denoted as Honeyindex+) and Superword (denoted as Superword+). Particularly, we reveal that both Honeyindex+ and Superword+ separate the relationship between usernames and passwords, while they are of different parameter settings under the same framework, and can be mutually converted. This framework enables the design of a generic attack theory to evaluate the security of various internally sampled honeyword schemes.
- **Rigorous attack theories.** We propose generic Bayesian and intersection attack theories to evaluate the security of our generic framework that can lead to the optimal distinguishing strategy. Contrary to Wang et al.'s naive Bayesian attack [38] that infers a user's sweetwords, our Bayesian attack theory infers bijections between users and passwords to exploit the inherent correlation and facilitates distinguishing attacks. We also consider the case where $\mathcal{A}$ can obtain multiple snapshots of the password file, and devise the intersection attack theory.
- **Extensive evaluation.** Guided by our attack theories, we evaluate our attacks under $\mathcal{A}_1$∼$\mathcal{A}_3$ models, with each catering to attackers' varied guessing capabilities. For Honeyindex+ with k=40 sweetwords, we reveal that with only one guess per account, the basic $\mathcal{A}_1$ attacker (using leaked password datasets) success rate can reach 3.82%∼4.12%, breaking the ideal $1/k$ security. Also, within a total of 10^4 honeyword logins, $\mathcal{A}_1$ can identify 4.31%∼5.04% of all real passwords, and the success rate is 1.36∼1.62 times that of the prior naive Bayesian attack [38]. For more advanced $\mathcal{A}_2$ (resp. $\mathcal{A}_3$) who further exploits a user's PII (resp. sister password), their one-guess success rate can reach 17.8%∼38.6% (resp. 18.9%∼41.3%). They can also identify 18.6%∼44.8% (resp. 20.6%∼43.6%) of all real passwords within 10^4 honeyword logins. Similarly, for Superword+ with k=n=10^4,the $\mathcal{A}_1$'s success rate is increased by 6∼123 times compared with the ideal security, and $\mathcal{A}_2$∼$\mathcal{A}_3$ can identify 3.19%∼13.6% and 9.78%∼26.2% of real passwords, respectively. When multiple password files are further available, our intersection attack alone can identify 18.3%∼18.6% of real passwords in Honeyindex+.
- **Some insights.** We obtain a number of insights from our theories and experimental results. We reveal that, due to the Zipf distribution of passwords [34], it is inherently impossible to make internally sampled honeywords (by randomly selecting from other users' passwords) equally probable with the real password, so popular passwords are still easier to identify. Besides, we also show that deviating from our framework can bring critical security flaws, showing the importance of adhering to our framework.

II. Preliminaries

In this work, we mainly focus on internally sampled honeyword schemes (e.g., Honeyindex [13] and Superword [15]), where each user's honeywords are sampled from others' real passwords within the same website. We do not consider the List model [38] and the Bernoulli honeyword scheme [41] as internally sampled, because they require external password datasets and honeyword generation techniques (HGTs). Specifically, the List model [38] directly utilizes passwords in an external dataset as honeywords. For the Bernoulli honeyword scheme [41], each user's honeywords are independently selected from an externally generated password model. Therefore, it is structurally similar to externally generated ones (e.g., [10], [18], [35], [38], [41]), and can be analyzed in the same way as Wang et al.'s naive Bayesian attack [38]. Hence, it is reasonable to treat the Bernoulli honeywords [41] as externally generated.

As shown in Fig. 1, we consider four entities in our system model: the user u, the authentication server $\mathcal{S}$, the honeychecker $\mathcal{HC}$, and the attacker $\mathcal{A}$. Like prior studies [18], [35], [38], [41], we assume that $\mathcal{HC}$ itself is intact and only communicates with $\mathcal{S}$ via a secure channel. In this way, the server $\mathcal{S}$ and the honeychecker $\mathcal{HC}$ form a honeyword scheme, and are resistant to offline guessing attacks even if $\mathcal{S}$ is compromised. In general, the system works as follows. First, the user registers an account (u, pw_u) at the server $\mathcal{S}$. Then, the sweetword list $\mathbf{swl}_u$ consisting of $k-1$ decoy passwords (called honeywords) and u's real password pw_u (e.g., k=40 [38]) is constructed by $\mathcal{S}$, and the index (denoted as id_u) of pw_u in $\mathbf{swl}_u$ is stored on $\mathcal{HC}$. Next, when the user u or the attacker $\mathcal{A}$ logs in with pw_u^*, $\mathcal{HC}$ checks if the index id_u^* of the submitted sweetword has $id_u^* \neq id_u$, signaling a potential password file leakage.

A. Two existing internally sampled honeyword schemes

To our knowledge, there are two internally sampled honeyword schemes, i.e., Honeyindex [13] and Superword [15]. For clarity, all symbols in these schemes are shown in Table I.

Honeyindex [13] is the first internally sampled honeyword scheme. In initialization, honeypots (i.e., fake accounts) are needed to sample honeywords for the first batch of users. In registration, the server $\mathcal{S}$ randomly assigns a unique index $id_u \in \{1, 2, ..., n\}$ to each user account, and computes the

TABLE I: Files stored in Honeyindex [13] and Superword [15].

Scheme	Password files on $\mathcal{S}^\dagger$		Files on $\mathcal{HC}$	
	F_1	F_2	F_3	F_4
Honeyindex [13]	$(u, \mathbf{sil}_u)$	$(id_{u'}, salt_{u'}\|\|H_{u'})$	(u, id_u)	–
Superword [15]	(u, r_u)	$(r_h,\ H_{u'})$	$(r_u, salt_u)$	(r_u, r_h)

† Entries in the F_1 and F_2 files need to be shuffled to obscure the relation between $(u, \mathbf{sil}_u)$ and $(id_{u'}, salt_{u'}||H_{u'})$. The notation $H_{u'}$ in the F_2 file means the salted password hash $H(pw_{u'}||salt_{u'})$, where u' means shuffled hashes of other users.

salted-hash H_u=$H(pw_u||salt_u)$. *In this manner, each index corresponds to a user and her password, and different indices can correspond to the same plaintext password (e.g.,* `iloveu`*) due to a highly skewed CDF-Zipf distribution in passwords [34].* Then, for each u, $\mathcal{S}$ uniformly samples $k-1$ indices from others, i.e., $\{1, 2, \cdots, n\}\backslash\{id_u\}$, and uses their passwords as u's honeywords. These k–1 indices (resp. honeywords) along with the real one form u's sweetindex list $\mathbf{sil}_u$ (resp. sweetword list $\mathbf{swl}_u$). In this design, an index can occur in multiple users' sweetindex lists, so sweetindices of different users are correlated. Next, the entries $(u, \mathbf{sil}_u)$ and $(id_u, salt_u||H(pw_u||salt_u))$ are randomly inserted into the F_1 and F_2 files, respectively, and the secret (u, id_u) is stored in F_3 on $\mathcal{HC}$.

In the above design, upon u's login with pw_u^*, $\mathcal{S}$ searches u's hashes in the sweetword list $\mathbf{swl}_u$ that matches the hash of the submitted pw_u^*; If not, the login fails; Otherwise, $\mathcal{S}$ obtains the index id_u^*, sends (u, id_u^*) to $\mathcal{HC}$, and $\mathcal{HC}$ checks whether id_u^*=id_u can hold. An equality means logging in with the correct password, i.e., pw_u^*=pw_u. An excessive number (e.g., 10^4) of $id_u^* \neq id_u$ suggests a password file leakage.

Superword [15] is another internally sampled honeyword scheme where honeypots are needed to enable leakage detection. In registration, the authentication server $\mathcal{S}$ randomly assigns a unique username index r_u to u, hashes pw_u with $salt_u$, and randomly assigns a unique hash index r_h to $H(pw_u||salt_u)$. The entries (u, r_u) and $(r_h, salt_u||H(pw_u||salt_u))$ are randomly inserted into the F_1 and F_2 files, respectively, in a way that *for any user, all other users' passwords serve as honeywords.* The honeychecker $\mathcal{HC}$ stores secret $(r_u, salt_u)$ and (r_u, r_h) in F_3 and F_4 files, respectively. *Whether an account is a honeypot is securely stored/marked in F_4 on $\mathcal{HC}$.* In this design, upon u's login with pw_u^*, $\mathcal{S}$ firstly queries $\mathcal{HC}$ for the user's salt by providing r_u; Then, $\mathcal{S}$ matches the hash of the submitted pw_u^* in all users' password hashes: If not, the login fails; Otherwise, $\mathcal{S}$ obtains the corresponding hash index r_h^* and sends (r_u, r_h^*) to $\mathcal{HC}$. Regardless of whether $\mathcal{HC}$ can find (r_u, r_h^*)=(r_u, r_h), $\mathcal{HC}$ will check if (r_u, r_h) corresponds to a honeypot. Triggering honeypots multiple times suggests a password file leakage.

B. Security model

Honeyword distinguishing attacker. The security goal of a honeyword scheme is to make the generated k–1 honeywords as indistinguishable from u's real password pw_u as possible. This goal is defined against the honeyword distinguishing attacker $\mathcal{A}$ (see Fig. 1) who can obtain and offline recover all salted-hashed sweetwords and regard the authentication server $\mathcal{S}$ as the online verification oracle. $\mathcal{A}$ will be detected by the honeychecker $\mathcal{HC}$ if she uses honeywords in logins: (1) If the number of honeyword login attempts exceeds the per-user threshold $\mathcal{T}_1$ (e.g., $\mathcal{T}_1$=3), the user will be alarmed; (2) If the number of honeyword login attempts exceeds the system-wide threshold $\mathcal{T}_2$ (e.g., $\mathcal{T}_2$=10^4), *all* users will be alarmed by password file leakage. In this case, to avoid being detected, $\mathcal{A}$ needs to devise a guessing strategy to identify the sweetword that is most likely to be the user's real password for online verification, and minimize the number of honeyword login attempts.

TABLE II: Distinguishing attackers' capabilities explored in this work.

Attacker types	Public info†	PII‡	Sister PW#	Studies consider Single PW file§	Studies consider Multiple PW files§	New in internally sampled scheme*
$\mathcal{A}_1$	✓	×	×	[10], [13], [18], [35] [15], [38], [41], [45]	[42]	✓
$\mathcal{A}_2$	✓	✓	×	[10], [35], [38], [41], [45]	None	✓
$\mathcal{A}_3$	✓	×	✓	[17]	None	✓

† Typical public info includes the various leaked password datasets and all the cryptographic algorithms (e.g., the honeyword generation techniques).
‡ Personally identifiable information (PII) includes name, birthday, etc.
# We mainly consider sister passwords generated by modifying existing passwords, rather than those generated from exact reuse.
§ After compromising the authentication server, the attacker can obtain a single or multiple password files. Wang-Reiter [42] consider the attacker's capability of obtaining multiple PW files, but they did not explore how public information like leaked datasets can facilitate distinguishing attacks.
* This column records whether this work is the first one (to the best of our knowledge) to explore $\mathcal{A}_1 \sim \mathcal{A}_3$ in internally sampled honeyword schemes.

Attackers' capabilities. As shown in Table II, we assume that the attacker can passively compromise the authentication server, obtain the password file, and know the associated hash algorithm as well as how honeywords are generated/sampled. This is consistent with attack models in Honeyindex [13] and Superword [15], and has been studied in prior studies [10], [18], [35], [38], [41], [45]. Specifically, depending on varied guessing capabilities, the basic attacker $\mathcal{A}_1$ can use publicly leaked password datasets. More advanced $\mathcal{A}_2$ can access users' personally identifiable information (i.e., PII, like name and birthday), and $\mathcal{A}_3$ can use sister passwords (i.e., passwords of the same users at other sites). Such attackers are realistic since users' PII can be obtained from various large-scale PII leaks (e.g., social media [6]), and sister passwords can be obtained from prior leakages [47]. We further assume that $\mathcal{A}_1 \sim \mathcal{A}_3$ can compromise the server multiple times (see [19], [27], [46]) and may obtain different snapshots of the password file, where a user's sweetindices are different. *To the best of our knowledge, we are the first to consider these realistic attacker models in the security analysis of internally sampled honeyword schemes.*

DoS attacker. Besides distinguishing attacks, denial-of-service (DoS) attacks that deliberately trigger false breach alarms are of practical concerns. The attacker $\mathcal{A}$ aims not to identify a user's real password, *but to deliberately trigger false breach alarms.* As in prior studies [13], [38], [41], we assume that DoS attackers can register multiple accounts (e.g., 10^4) with their passwords, and log in with these passwords on ordinary users to trigger false alarms. This is reasonable, as DoS attackers do not need to obtain users' plaintext passwords to trigger false alarms, and a user's honeywords are randomly sampled from others' passwords in internally sampled honeyword schemes.

Other attacks. As discussed in [3], [18], [35], [38], other attacks such as multi-system intersection (exploiting password reuse) and memory extraction attacks are also practical. Multi-system intersection attacks can be mitigated by Wang and Reiter's cryptographic approach [33]. Memory extraction

attacks obtain passwords through non-cryptographic means and are therefore outside the scope of this work.

C. Evaluation metrics

Flatness graph [35] plots the average success rate y of the attacker $\mathcal{A}$ to identify the real password within $1{\leq}x{\leq}k$ sweetword login attempts *per account*. Suppose that N_x is the number of an attacker's cracked accounts after x online guesses. A point (x, y) on a curve means that the real password is identified with a probability y (on average) in the first x sweetword attempts. Here, $y{=}N_x/n$ and n is the number of all accounts, *so all users share one flatness graph.* Consequently, the point $(x{=}1, y)$ denotes the ϵ-flat given in [18], i.e., $\epsilon{=}y|_{x=1}$, and an ideal honeyword generation technique (HGT) should have $\epsilon{=}1/k$. Per-user alarms will be triggered when $x{\geq}\mathcal{T}_1$ (e.g., 3), and this metric measures an attacker's per-user distinguishing capability. Since different users' sweetwords are correlated, $\mathcal{A}$ can use feedback in distinguishing others' passwords to more efficiently identify the target one. This makes the flatness graph plotted in a way different from prior studies [35], [38], and we will explain in detail in Sec. IV-C.
Success-number graph [35] plots the total number of successful login attempts y when $\mathcal{A}$ tries $x{\leq}\mathcal{T}_2$ (e.g., 10^4) failed logins (less than $\mathcal{T}_1$ per account), regardless of whose honeywords are attempted. A point (x, y) on a curve means that y real passwords are successfully distinguished with x honeyword login attempts. For an ideal HGT (i.e., $\epsilon{=}1/k$), each sweetword is equally probable to be the real one, so no particular password is easier to distinguish. This metric measures an attacker's system-wide distinguishing capability. Similarly, in DoS attacks, (x, y) means that honeywords of y accounts reach the $\mathcal{T}_1$ threshold with x honeyword login attempts.

III. Generic Framework of Internally Sampled Honeyword Schemes

We first reveal two critical security and two critical usability flaws in existing Honeyindex [13] and Superword [15]. Then, we provide our generic framework for sound internally sampled schemes. Finally, we instantiate Honeyindex+ and Superword+ under our framework, and reveal how they address these flaws.

A. Design flaws in existing schemes

For Honeyindex [13], its design entails a usability flaw that can deny login attempts with correct passwords. Specifically, since passwords are Zipf distributed [34], duplicate honeywords equal to the real password pw_u can be sampled, though each of them has a unique index. To our knowledge, Honeyindex [13] did not consider such impacts: The sweetindex of *one* (not all) of the matched sweetwords is sent to the honeychecker $\mathcal{HC}$ to check whether it matches the user u's index id_u assigned during registration (see Sec. II-A). In this case, even if the user u logs in with the real password pw_u, the submitted sweetindex id_u^* may not match id_u stored on $\mathcal{HC}$, i.e., $id_u^*{\neq}id_u$. Thus, u is unable to log in, and causes false authentication denials. As a result, breach alarms may also be triggered by legitimate users' logins (e.g., when the number of false authentication denials exceeds $\mathcal{T}_2$), making leakage detection unreliable.

For Superword [15], unlike other schemes (see [13], [18], [35], [38], [41]) that store users' salts on the server $\mathcal{S}$, it stores salts on the honeychecker $\mathcal{HC}$ (see Sec. II-A), and relies solely on honeypots for leakage detection. We show the two security and one usability flaws entailed by its design as follows.

- The first security flaw of Superword is that it cannot detect the leakage of a specific user's password. Since user-specific salts are stored on $\mathcal{HC}$, when a honeyword is tried, only the user's salt is sent to $\mathcal{S}$ to compute the salted hash, which cannot match any stored hash. As a result, $\mathcal{HC}$ does not trigger an alarm. This differs from Honeyindex [13] and externally generated honeyword schemes [18], [35], [38], [41], where a per-user alarm is triggered once the number of failed logins exceeds $\mathcal{T}_1$ (e.g., $\mathcal{T}_1$=3).
- The second security flaw is that Superword cannot simultaneously achieve high detection and low false-alarm rates. Unlike other schemes where alarms are triggered mainly by honeywords, Superword [15] triggers alarms only by honeypots, regardless of password content. Increasing the number of honeypots improves detection rates but also raises false-alarm rates from DoS attacks. Consequently, mitigating DoS attacks in Superword is difficult without reducing leakage detection rates.
- The usability flaw of Superword is a single point of failure when $\mathcal{HC}$ storing salts fails (e.g., software malfunctions), $\mathcal{S}$ cannot compute salted hashes and verify whether a submitted password pw_u^* matches any sweetword. Consequently, even correct passwords cannot log in. In contrast, other schemes do not rely on $\mathcal{HC}$ for verification, allowing users to log in normally using Juels and Rivest's approach that temporarily allows logging in with sweetwords [18].

Besides, like other honeyword schemes (e.g., [18], [35], [38], [41]), Superword [15] cannot achieve the claimed ideal (i.e., $1/n$) security regardless of where salts are stored. Since a user can always log into Superword with the correct password [15], a popular password (e.g., `iloveu`) will be used with a rate greater than $1/n$, thereby breaking the $1/n$ ideal security.

B. Workflow of our generic framework

In general, a sound internally sampled honeyword scheme should: (1) sample honeywords from users' real passwords within the system, (2) allow a user to log in with the correct password, and (3) enable both per-user and system-wide leakage detections. In such a design, for each user u, when the real password pw_u is submitted to the authentication server $\mathcal{S}$, the login succeeds; When $sw{\neq}pw_u$ is submitted (mostly by attackers), the honeychecker $\mathcal{HC}$ can identify such a threat.

To achieve these goals, the tuple $(u, salt_u||H(pw_u||salt_u))$ *needs to be separately stored, and only the correct* pw_u *can restore it*, which can be done by the command $\mathsf{Split}(u, pw_u)$. More precisely, $\mathsf{Split}(u, pw_u)$ splits the tuple (u, pw_u) into user-related (u, ID_u) and password-related $(ID_u, salt_u||H(pw_u||salt_u))$, where ID_u is u's index under a given indexing method (e.g., alphabetical order). The set of all ID_u is denoted as $\mathcal{ID}$, and each ID_u associates with a user and the user's password. Since user authentication is conducted on $\mathcal{S}$ and requires salts, the password-related $(ID_u, salt_u||$

$H(pw_u||salt_u)$) needs to be stored on $\mathcal{S}$. Consequently, the user-related (u, ID_u) needs to be secretly stored on $\mathcal{HC}$.

Next, for each user u, the server $\mathcal{S}$ samples $k-1$ users' indices from $\mathcal{ID}\backslash\{ID_u\}$ via the command $\mathsf{Sample}(k, u)$, and the associated passwords are used as u's honeywords. Along with the real index (resp., password), the k elements collectively form u's sweetindex list $\mathbf{sil}_u$ (resp. sweetword list $\mathbf{swl}_u$). In this way, for any $k<n$, the server $\mathcal{S}$ maintains the associated sweetindex list $\mathbf{sil}_u$ of length k, and the tuple $(u, \mathbf{sil}_u)$ is stored on $\mathcal{S}$. When k=n, $\mathcal{S}$ treats all other users' indices as decoys, so all users share the same sweetindex/sweetword list.

We now show how to store (u, ID_u) on $\mathcal{HC}$. Since each (salted-hashed) sweetword is a user's real password, ID_u should connect the user's indices and her/his password $pw_u \in \mathcal{PW}$, i.e., *a chain of indices from the password hash to its user.* If the chain is of length l, that is, there are l indices in the chain except the hash stored on $\mathcal{S}$, $\mathcal{HC}$ needs to separately store l files, with each index appearing in two adjacent files. In this paper, we mainly consider l=1 and 2, and larger l values are similar. We now present the detailed workflow.

Initialization. The server $\mathcal{S}$ registers honeypots (i.e., fake accounts) to offer a pool of passwords to sample honeywords. To prevent attackers from identifying honeypots after compromising $\mathcal{S}$, the mark recording whether an account is a honeypot is stored on the honeychecker $\mathcal{HC}$, e.g., (u, ID_u) is marked with 1 if u is a honeypot and 0 otherwise. For simplicity, honeypots and real users are collectively denoted as users.

Registration. The server $\mathcal{S}$ executes commands $\mathsf{Split}(u, pw_u)$ and $\mathsf{Sample}(k, u)$ after u logs in with pw_u. Specifically, $\mathsf{Split}(u, pw_u)$ outputs $(ID_u, salt_u||H(pw_u||salt_u))$ and (u, ID_u), which are stored on $\mathcal{HC}$ and $\mathcal{S}$, respectively. The command $\mathsf{Sample}(k, u)$ then uniformly samples $k-1$ distinct indices from other users' indices to construct the sweetindex list $\mathbf{sil}_u$. Due to the popular passwords, different sweetindices may associate with the same plaintext sweetword.

Login. When u logs in with pw_u^*, the server $\mathcal{S}$ first computes $H(salt_{u'}||pw_u^*)$ with $salt_{u'}$ being the salt of each sweetword, then checks whether it matches any stored hash. If no match can be found, the login fails. Otherwise, the sweetindex set $\mathcal{I}_u$=$\{ID_u^1, ID_u^2, ..., ID_u^d\} \subseteq \mathbf{sil}_u$ (with $1 \le d \le k$) of *all* matched sweetwords is sent to the honeychecker $\mathcal{HC}$ by the command $\mathsf{Check}(u, \mathcal{I}_u)$ to check whether one can match u's real index ID_u stored on $\mathcal{HC}$, If so, the login succeeds.

Detection. The alarm can be triggered if: (1) u is a honeypot, (2) none of the sweetindices in $\mathcal{I}_u$ match u's real index stored on $\mathcal{HC}$. When the number of failed login attempts for each user exceeds $\mathcal{T}_1$ (e.g., 3), per-user alarms will be triggered. Accordingly, if the number of failed login attempts for all users exceeds $\mathcal{T}_2$ (e.g., 10^4), system-wide alarms will be triggered.

Regular updates. To resist the peeling-onion style attack [35] that exploits the domino effect of an isolated sweetindex (i.e., onion), as recommended in [13], the server $\mathcal{S}$ regularly executes $\mathsf{Sample}(k, u)$ to resample honeywords for all users.

Our generic framework as illustrated in Fig. 1 addresses three key issues. First, by storing salts on the server $\mathcal{S}$, it avoids the first three design flaws in Superword [15] entailed by storing user-specific salts on $\mathcal{HC}$. Second, the command $\mathsf{Check}(u, \mathcal{I}_u)$ in our framework allows u to log in with pw_u regardless of duplicate sweetwords, preventing false authentication denials. Third, compared with storing all sweetindices of a password on $\mathcal{HC}$, our framework reduces the storage cost of $\mathcal{HC}$, and maintains Juels-Rivest's minimalist principle [18].

C. Instantiated schemes

We denote Honeyindex [13] and Superword [15] variants instantiated under our framework as Honeyindex+ and Superword+, and show their key commands as follows.

- $\mathsf{Split}(u, pw_u)$. In Honeyindex+ and Honeyindex, the indexing method directly records the index of each user's password, i.e., ID_u=$id_u \in \{1, 2, \cdots, n\}$, and (u, id_u) is stored in the F_3 file on the honeychecker $\mathcal{HC}$ as our framework regulates. This corresponds to the shortest chain of indices with the length l=1. In Superword+, ID_u=(r_u, r_h), where r_u and r_h are random indices of the user and its hash, respectively. By design, (u, r_u) and (r_u, r_h) are separately stored in F_3 and F_4 respectively on $\mathcal{HC}$, and $(r_h, salt_u||H(pw_u||salt_u)$ is stored in F_2 on $\mathcal{S}$. This corresponds to the case of l=2.
- $\mathsf{Sample}(k, u)$. In Honeyindex+ and Honeyindex, the number of sampled indices is k–1$<n$–1, so users have different sweetindex lists as well as associated sweetword lists. In Superword+ and Superword, since k=n, all users share the same sweetindex/sweetword list in F_2.
- $\mathsf{Check}(u, \{ID_u^1, ID_u^2, ..., ID_u^d\})$. In Honeyindex+, $\mathcal{HC}$ checks whether $id_u \in \{id_u^1, id_u^2, ..., id_u^d\}$ due to ID_u=id_u. If so, the login succeeds. In Superword+, $\mathcal{HC}$ checks whether $(r_u, r_h) \in \{(r_u, r_h^1), ..., (r_u, r_h^d)\}$ due to ID_u=(r_u, r_h). In Honeyindex and Superword, this command is $\mathsf{Check}(u, id_u^*)$ and $\mathsf{Check}(u, (r_u, r_h^*))$, since only one sweetindex of matched sweetwords is sent to $\mathcal{HC}$.

The above content reveals the two major differences between Honeyindex+ and Superword+. First, ID_u under different indexing methods leads to different storage configurations in $\mathcal{HC}$ and executions of $\mathsf{Check}(u, \{ID_u^1, ..., ID_u^d\})$. Second, in Honeyindex+, k–1 users' passwords are used as honeywords, so users' sweetword lists are different. As a corollary, different internally sampled honeyword schemes are mainly results of different instantiations of the above key commands.

These instantiations also corroborate how Honeyindex+ and Superword+ avoid design flaws in the original schemes [13], [15]. For Honeyindex+, u can always log in with pw_u since sweetindices of all sweetwords matching the submitted pw_u^* are sent to $\mathcal{HC}$ for verification, and the real index must be in it if pw_u^*=pw_u. For Superword+, design flaws entailed by storing salts on $\mathcal{HC}$ can be avoided. Unless stated otherwise, we mainly explore Honeyindex+ and Superword+.

IV. Our Attack Theories

In this section, we first present our Bayesian attack theory to evaluate the security of our generic framework, then introduce the intersection attack theory when $\mathcal{A}$ can obtain multiple snapshots of password files due to the recommended regular update mechanism (i.e., periodically resampling honeywords [13]). Finally, we also explore the impacts of DoS attacks.

A. Our Bayesian attack theory

The goal of the distinguishing attacker $\mathcal{A}$ is to form an optimal online guessing/verification order to online verify sweetwords as efficiently as possible before triggering alarms (e.g., per-user $\mathcal{T}_1$=3 and system-wide $\mathcal{T}_2$=10^4, see Sec. II-C). To study this, we propose a Bayesian attack theory to calculate the probability that a sweetword is the real password with prior knowledge. Unlike externally generated honeywords (see [18], [35], [38], [41], [45]) that are independent across users, in internally sampled honeyword schemes, a user's real password is others' honeywords, forming a bijection (i.e., one-to-one map) between users and passwords. This relationship enables $\mathcal{A}$ to conduct more efficient honeyword distinguishing attacks than random guesses. For instance, suppose the sweetword lists of u_1, u_2 and u_3 are $(pw_1, pw_3), (pw_2, pw_3)$ and (pw_3, pw_2). If pw_3 is u_1's real password, pw_3 must be a honeyword of both u_2 and u_3, so pw_2 must be the real password for both u_2 and u_3, which contradicts the definition of bijection.

The above example illustrates that when $\mathcal{A}$ attacks internally sampled honeyword schemes, treating one sweetword as a user's real one impacts identifying others' passwords. *Hence, it is $\mathcal{A}$'s optimal strategy to exploit the relationship between identified user-sweetword pairs to maximize the success rate.* To formally depict how a user u corresponds to a password pw, we introduce a bijection $\phi : \mathcal{U} \rightarrow \mathcal{ID}$ from the user set $\mathcal{U}$ to the index set $\mathcal{ID}$, and a surjection $\nu : \mathcal{ID} \rightarrow \mathcal{PW}$ from $\mathcal{ID}$ to the password set $\mathcal{PW}$, with $\phi(u)$=$ID_{u'}$ (not necessarily u's ID_u) and $\nu(ID_u)$=pw_u. Since passwords are Zipf distributed [34], a password may be adopted by multiple users, so ν may not be a bijection. We mainly focus on ϕ since ν is fixed for each internally sampled scheme. To explore the relationship between users and their sweetindices that $\mathcal{A}$ can exploit after compromising $\mathcal{S}$, we provide the following definitions.

Definition 1: A bijection ϕ^+ is valid if it maps each user u to one of her sweetindices, i.e., there are $\phi^+(u) \in \mathbf{sil}_u$ and thus $\nu(\phi^+(u)) \in \mathbf{swl}_u \ \forall u \in \mathcal{U}$. Let Φ^+ denote the set of all valid bijections. A valid bijection $\phi^* \in \Phi^+$ is real if it maps each u to one of its sweetindices whose associated sweetwords are pw_u, i.e., $\nu(\phi^*(u))$=$pw_u \ \forall u \in \mathcal{U}$. Other valid bijections are decoys.

We explain the above definition. A valid bijection ϕ^+ reveals whether a password is a user's sweetword, and the real bijection ϕ^* records which sweetword is real. As assumed in prior studies [13], [35], [38], when the authentication server $\mathcal{S}$ is compromised, both the F_1 and F_2 files (see Sec. III-C) storing sweetindices and their corresponding sweetwords (in salted-hashes) are leaked. Hence, $\mathcal{A}$ can know valid bijections, but not which is real. *As a consequence, $\mathcal{A}$'s task of identifying the real passwords is converted to identifying the real bijection from valid bijections, whereas invalid bijections (which map a user to others' sweetwords) are unnecessary.* To the best of our knowledge, we are the first to consider inferring bijections rather than sweetwords in Bayesian estimation.

To infer valid bijections, the probability $\Pr_{\Phi^+}[\phi^+]$ of selecting a particular valid bijection $\phi^+ \in \Phi^+$, which is essentially the posterior probability $\Pr_{\Phi^+}[\phi^+|F_1, F_2]$, needs to be quantified. Since ϕ^+ is determined by how it maps each u to her sweetindex and ν is fixed, the probability of selecting ϕ^+ can be measured by that of selecting $sw \in \mathbf{swl}_u$ (allowing duplicates) satisfying $v(\phi^+(u))$=sw in each $\mathbf{swl}_u$. Formally, there is

$$w(u, \phi^+(u)) = \frac{\Pr_{\mathrm{PW}}[v(\phi^+(u))]}{\sum_{sw \in \mathbf{swl}_u} \Pr_{\mathrm{PW}}[sw]}, \tag{1}$$

where $w(u, \phi^+(u))$=$w(u, \phi^+(u)|F_1, F_2)$ denotes the weight function, and $\Pr_{\mathrm{PW}}[\nu(\phi^+(u))]$=$\Pr_{\mathrm{PW}}[\nu(\phi^+(u))|F_1, F_2]$ is the probability that u's sweetword $\nu(\phi^+(u))$ is selected when the F_1 and F_2 files are available. Upon these, $\Pr_{\Phi^+}[\phi^+] \propto \prod_{u \in \mathcal{U}} w(u, \phi^+(u)) \propto \prod_{u \in \mathcal{U}} \Pr_{\mathrm{PW}}[\nu(\phi^+(u))]$, and we obtain

$$\Pr_{\Phi^+}[\phi^+] = \frac{\prod_{u \in \mathcal{U}} w(u, \phi^+(u))}{\sum_{\phi^+ \in \Phi^+} \prod_{u \in \mathcal{U}} w(u, \phi^+(u))}. \tag{2}$$

Based on these premises, we can obtain the key Theorem 1 about the attacker's guessing strategy as follows.

Theorem 1: Let ϕ^* denote the event of selecting $\phi^* \in \Phi^+$ as the user's real bijection, and $\{\mathbf{swl}_u | u \in \mathcal{U}\}$ denote all users' sweetword lists. Once the authentication server $\mathcal{S}$ is compromised and the F_1 and F_2 files are available, the attacker $\mathcal{A}$ should try in descending order of the following probability

$$\Pr_{\Phi^+}[\phi^* | \{\mathbf{swl}_u | u \in \mathcal{U}\}] = \frac{\Pr_{\Phi^+}[\phi^*]}{\sum_{\phi^+ \in \Phi^+} \Pr_{\Phi^+}[\phi^+]}, \tag{3}$$

under the assumption that $\{\mathbf{swl}_u | u \in \mathcal{U}\}$ are independently sampled and a user's real password depends on $\phi^* \in \Phi^+$.

The proof is shown in Appendix A. This theorem reveals that in identifying real passwords, the attacker $\mathcal{A}$ can leverage sweetword lists of *all* users by inferring the Bayesian posterior probability of ϕ^+, instead of only a user's sweetword list as in Wang et al.'s Bayesian attack (i.e., the naive Bayesian attack, see Theorem 2) [38]. Thus, our Bayesian attack theory can be more effective. The details are shown in Sec. IV-B.

More concretely, if internally sampled honeyword schemes aim to achieve the ideal flatness/security, each valid bijection in Φ^+ should have an equal probability of being real. In other words, given the real bijection ϕ^*, internally sampled honeyword schemes need to produce users' sweetword lists such that each valid bijection should have an *equal* probability of being the real one ϕ^*. However, this is inherently impossible since user-chosen passwords follow CDF-Zipf distributions [34]: For passwords ranked in descending order of probability, there is 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 r-th popular password, and $s \in [0.15, 0.30]$ and $C \in [0.001, 0.1]$ are constants; As p_r decreases sharply with r when r is small, i.e., when a user's password is popular, the probabilities of sweetwords in $\mathbf{swl}_u$ vary largely, so does $\Pr_{\Phi^+}[\phi^*]$. *This means that internally sampled honeyword schemes cannot achieve the claimed ideal flatness/security.*

The above results assume that both F_1 and F_2 are leaked. For completeness, we briefly analyze three related cases. First, if neither F_1 nor F_2 is leaked, a bijection ϕ must be treated as a random permutation over $\{1, 2, \ldots, n\}$, since $\mathcal{A}$ cannot determine which maps are valid. Second, if only F_1 is leaked, ϕ^+ is determined by how it maps each user to her uniformly distributed sweetindices, there exists $w(u, \phi^+(u))$= $w(u, \phi^+(u)|F_1) \equiv 1/k$ and $\Pr_{\Phi^+}[\phi^+]$=$\Pr_{\Phi^+}[\phi^+|F_1] \equiv 1/k^n$ for all $\phi^+ \in \Phi^+$, i.e., simplifications of Eqs. 1 and 2. Third, if only F_2 is leaked, how a user associates her sweetindices is

unknown, ϕ^+ becomes a random permutation. *Thus, unless stated otherwise, we assume that both F_1 and F_2 are leaked.*

We also investigate how user-specific auxiliary information (denoted as X_u) can facilitate attacks. As shown in Table II, all $\mathcal{A}_1 \sim \mathcal{A}_3$ can access publicly leaked password datasets, while more advanced $\mathcal{A}_2$ and $\mathcal{A}_3$ can further access users' personally identifiable information (i.e., X_u=PII$_u$) and sister passwords pw_u^s (i.e., X_u=pw_u^s) leaked at other websites, respectively. In this case, as natural extensions of Eqs. 1 and 2, we have

$$\begin{cases} w(u,\phi^+(u)|\mathrm{X}_u) = \dfrac{\mathrm{Pr}_{\mathrm{PW}}[\nu(\phi^+(u))|\mathrm{X}_u]}{\sum_{sw\in\mathbf{swl}_u}\mathrm{Pr}_{\mathrm{PW}}[sw|\mathrm{X}_u]} \\ \mathrm{Pr}_{\Phi^+}[\phi^+|\mathrm{X}_u] = \dfrac{\prod_{u\in\mathcal{U}} w(u,\phi^+(u)|\mathrm{X}_u)}{\sum_{\phi^+\in\Phi^+}\prod_{u\in U} w(u,\phi^+(u)|\mathrm{X}_u)}. \end{cases} \tag{4}$$

Accordingly, as an extension of Theorem 1, $\mathcal{A}$ should try in descending order of the probability $\mathrm{Pr}_{\Phi^+}[\phi^*|\{\mathbf{swl}_u|u\in\mathcal{U},\mathrm{X}_u\}]$=$\frac{\mathrm{Pr}_{\Phi^+}[\phi^*|\mathrm{X}_u]}{\sum_{\phi^+\in\Phi^+}\mathrm{Pr}_{\Phi^+}[\phi^+|\mathrm{X}_u]}$. Since X_u is often highly correlated with the user's real password pw_u [37], $w(u,\phi^+(u)|\mathrm{X}_u)$ is likely to be larger when ϕ^+ is the real bijection, i.e., $\phi^+(u)$=pw_u. *This reveals the inherent limitation of internally sampled schemes, as honeywords sampled from other users are unlikely to be correlated with X_u.* We will elaborate on this in Sec. V-D.

B. Comparison with existing attacks

Guo et al. [15] claimed their matching attack against Honeyindex [13] could achieve 100% success, assuming salts are stored with usernames. However, salts are stored with password hashes, making their attack impractical. Erguler [13] ignored correlations across users' sweetwords and duplicate sweetwords sampled from highly skewed CDF-Zipf distributions [34], and thus falsely concluded that Honeyindex could achieve ideal security [13]. We use Theorem 2 to reveal how the naive Bayesian attack causes the false $1/k$ ideal security.

Theorem 2 ([38]): Let $\mathrm{Pr}[pw]$ denote the probability of selecting pw_u as the real password, and $\mathrm{Pr}[sw|pw]$ denote the probability of selecting sw as the honeyword when pw is the real password. Then the attacker $\mathcal{A}$ should guess the user u's sweetwords in descending order of the posterior probability

$$\mathrm{Pr}[pw|\mathbf{swl}_u]=\frac{\mathrm{Pr}[pw]\prod_{sw\in\mathbf{swl}_u\setminus\{pw\}}\mathrm{Pr}[sw|pw]}{\sum_{pw'\in\mathbf{swl}_u}\mathrm{Pr}[pw]\prod_{sw\in\mathbf{swl}_u\setminus\{pw'\}}\mathrm{Pr}[sw|pw']}, \tag{5}$$

under the condition that $\{sw\in\mathbf{swl}_u\setminus\{pw\}\}$ are mutually independent when u chooses pw as her real password.

Firstly, the difference between Theorems 1 and 2 explains why our Bayesian attack is more effective. Specifically, Theorem 2 [38] ignores correlations among users' sweetwords: Even if a user's real password has been identified, once it appears in others' sweetword lists, it will still be treated as unknown. In contrast, our Theorem 1 exploits the feedback from identified passwords: If $\mathcal{A}$ has distinguished u's real password pw_u, she only needs to focus on bijections satisfying $\nu(\phi^+(u))$=pw_u in future attacks, thus enhancing the attack efficiency.

Secondly, it is necessary to treat duplicate sweetwords as different ones to achieve the $1/k$ security. In Honeyindex [13], a user's honeywords are independently sampled, so for any $sw\in\mathbf{swl}_u\setminus\{pw_1,pw_2\}$, we have $\mathrm{Pr}[sw|pw_1]$=$\mathrm{Pr}[sw|pw_2]$, thus $\mathrm{Pr}[sw]$=$\mathrm{Pr}[sw|pw_u]$ for all $sw\in\mathbf{swl}_u\setminus\{pw_u\}$. Moreover, since each sweetindex is unique and only one sweetindex of matched sweetwords is sent to the honeychecker $\mathcal{HC}$ for verification, duplicate sweetwords are treated as distinct elements (see Sec. II-A). Thus, Eq. 5 can be simplified as

$$\mathrm{Pr}[pw|\mathbf{swl}_u]=\frac{\prod_{sw\in\mathbf{swl}_u}\mathrm{Pr}[sw]}{\sum_{sw\in\mathbf{swl}_u}\prod_{sw\in\mathbf{swl}_u}\mathrm{Pr}[sw]}=\frac{1}{k} \tag{6}$$

indicating the claimed $1/k$ security in [13]. In contrast, logins with the user's correct password can always succeed in our framework (see Sec. III-B). Since passwords are Zipf distributed [34], popular passwords are more likely to be chosen as users' honeywords, resulting in duplicate sweetwords. When $\mathcal{A}$ tries such a sweetword pw, Eq. 6 recording $\mathcal{A}$'s success rate becomes $\mathrm{Pr}[pw|\mathbf{swl}_u]$=$|pw|/k$ with $|pw|\geq 2$, breaking the $1/k$ security. Therefore, the $1/k$ security is hard to achieve.

We next compare our Bayesian attack with Wang et al.'s peeling-onion style attack [35] claimed to achieve a 100% success rate against Honeyindex [13]. We reveal that our Bayesian attack works regardless of the regular update mechanism, whereas the peeling-onion attack cannot. At a high level, the peeling-onion attack [35] treats the isolated index that only appears in one user's sweetindex list (i.e., onion) as the user's real index, since only it has this feature. Then, the attack repetitively removes users' sweetindex lists containing the onion until all users' onions are identified. When no regular updates exist, the password of the last registered account becomes the onion, causing the above domino catastrophe.

However, if regular updates are enforced as recommended in [13], *finding an isolated index to perform the peeling-onion attack is almost impossible.* Formally, we give Eq. 7 as follows to quantify the probability of finding an isolated index ID with n user accounts and k sweetindices as

$$\mathrm{Pr}[ID]=\left(\frac{\binom{n-2}{k-1}}{\binom{n-1}{k-1}}\right)^{n-1}=\left(\frac{n-k}{n-1}\right)^{n-1}\approx e^{-k+1}, \tag{7}$$

where $\binom{n-1}{k-1}$ and $\binom{n-2}{k-1}$ are the numbers of all combinations of $k-1$ sweetindices when ID can and cannot exist, respectively. In this manner, $\mathrm{Pr}[ID]$ can be seen as the probability that the sweetindex lists of all $n-1$ users (except u) do not include ID, and the approximation holds due to $\lim_{n\to\infty}(1+1/n)^n$=$e$. For Honeyindex [13] with n=10^4, k=40, the probability of finding an isolated index is 1.07×10^{-17}. In contrast, regardless of regular updates, bijections between users and passwords exist, so our Bayesian attack theory can be constantly effective.

C. Efficient Bayesian attack algorithms

We mainly focus on the general case, and the cases with auxiliary information are similar. Before online guesses/verifications, $\mathcal{A}$ needs to determine the bijection ϕ^+ with the maximal probability (see Eq. 3), which can be converted to a combinatorial optimization problem. For the naive enumeration, however, the time complexity of searching all valid bijections is as inefficient as $O(n!)$. To find a more efficient solution, we formalize our problem as follows: For an $n\times n$ weight matrix W, where each row i represents the i-th user u_i, and each column j represents the j-th index ID_j,[1] and u_i selects the

[1]For Honeyindex+, we treat the rest of the $n-k$ indices as sweetindices, except that the weight of a user's selection of these sweetindex is always 0.

Algorithm 1: Greedy Search Approximation (GSA).

Input : Weight matrix $W_{n\times n}$ and sweetindex list $\mathbf{sil}_u$ for each user u.
Output : An approximately optimal matching ϕ^+ for verification.

begin
  Step 1: Row sorting
  for $u_i \in \mathcal{U}=\{u_1,\dots,u_n\}$ **do**
    $s\mathbf{sil}_{u_i} \leftarrow \mathsf{Sort}_j(\mathbf{sil}_{u_i}, w_{i,j})$; /* Sort $\mathbf{sil}_i$ in descending order by weight row w_{ij} in terms of $1\le j\le n$. */
    $i.c \leftarrow 0$, $i.v \leftarrow 0$, $pw_j = s\mathbf{sil}_{u_i}[i.c]$;
    if $\sum_{j'=1}^{n} w_{i,j'} = w_{i,j}$ **then**
      $i.v \leftarrow 1$; /* $s\mathbf{sil}_{u_i}[i.c]$ is verified as u_i's real password. */
  Step 2: Conflict handling
  for $u_i \in \mathcal{U}=\{u_1,\dots,u_n\}$ **do**
    $\phi^+(u_i) \leftarrow s\mathbf{sil}_{u_i}[i.c]$; /* Select the sweetindex with the highest weight in the sorted sweetindex list as the real one. */
    $ID_j = \mathbf{sil}_{u_i}[i.c]$;
    while $\exists u_i^*$ *s.t.* $\phi^+(u_i) = \phi^+(u_{i^*})$ **do**
      if $i^*.v = 1$ **then**
        if $i.c \ge k$ **then break**;
        $i.c$++, $\phi^+(u_i) = s\mathbf{sil}_{u_i}[i.c]$;
      if $w_{i^*,j} \ge w_{i,j}$ **then**
        if $i.c \ge k$ **then break**;
        $i.c$++, $\phi^+(u_i) \leftarrow s\mathbf{sil}_{u_i}[i.c]$; /* $\phi^+(u_{i^*})$ is more likely to be real for u_{i^*}. */
      else
        if $i^*.c \ge k$ **then break**;
        $i^*.c$++, $\phi^+(u_{i^*}) \leftarrow s\mathbf{sil}_{u_i^*}[i^*.c]$, $i \leftarrow i^*$;
  Step 3: Derive the approximately optimal matching
  Sort the matching results in descending order by $w(u,\phi^+(u))$, obtaining an approximately optimal matching ϕ^+ for verification;
  return *Approximately optimal matching* ϕ^+.;

password $\nu(ID_j)\in\mathcal{PW}$ with the weight w_{ij} to maximize the total weight. Formally, we can obtain the optimization task as:

$$\text{Maximize} \sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_{ij}, \text{ s.t } \begin{cases} \sum_{i=1}^{n} x_{ij} = 1, \\ \sum_{j=1}^{n} x_{ij} = 1, \\ x_{ij} = 1 \quad \text{if } u_i \text{ selects } sw_j, \\ x_{ij} = 0 \quad \text{otherwise,} \end{cases} \tag{8}$$

where the weight matrix w is initialized by our weight function $w(i,j)$ (see Eq. 1). The constraints $\sum_{i=1}^{n} x_{ij}=\sum_{j=1}^{n} x_{ij}=1$ hold since each sweetindex is the real index of only one user. The Hungarian algorithm [21], as an exact matching algorithm, can be used to solve our problem, but its complexity is as high as $O(n^4)$. To deal with this issue, we design an efficient approximate matching using a greedy search strategy to speed up searching for the nearly optimal solution.

Approximate matching. The proposed efficient approximate matching (i.e., Greedy Search Approximation, GSA) algorithm can prioritize matching with higher weights through greedy search while ensuring the corresponding success rate. As shown in Alg. 1, the GSA algorithm consists of two sub-processes: (1) "Row sorting" to facilitate finding an approximate maximum total weight using a greedy strategy; (2) "Conflict handling" to ensure the matching is as close to a bijection as possible. Details about the GSA algorithm are shown as follows:

- **Row sorting:** GSA first sorts a user u_i's sweetindex list $\mathbf{sil}_{u_i}$ in descending order of w_{ij} (denoted as $s\mathbf{sil}_{u_i}$) with $ID_j \in \mathbf{sil}_{u_i}$. Then, by setting u_i's current real password index as $i.c$=0, GSA prioritizes the sweetindex $sw=s\mathbf{sil}_{u_i}[i.c]$ with the highest weight as u_i's real password $\phi^+(u_i)$, i.e., $\phi^+(u_i) \leftarrow s\mathbf{sil}_{u_i}[i.c]$. Next, it checks if there is only one nonzero weight in a row: If so, $\nu(s\mathbf{sil}_{u_i}[i.c])$ is verified as u_i's real password and $i.v$=1.
- **Conflict handling:** Due to the sampling nature, an index can occur in multiple users' sweetindex lists. Thus, it can happen that both u_{i_1} and u_{i_2} select the same sweetindex ID_j, contradicting the constraint $\sum_{i=1}^{n} x_{ij}$=1, for j=1, ..., n cannot hold. To maintain this, the user who is already identified or has a higher weight remains her index as ID_j; For the user with a lower weight, GSA chooses the next available ID_j^* in $\mathbf{sil}_u$. For example, if $\phi^+(u_{i_1})=\phi^+(u_{i_2})=ID_j$ and $i_1.v$=1 or $w_{i_1,j}\ge w_{i_2,j}$, then GSA set $\phi^+(u_{i_1})=ID_j$ and $\phi^+(u_{i_2})=\mathbf{sil}_{i_2}[i_2.c+1]$.
- **Derive optimal matching:** This process sorts all user-password pairs in the matching result in descending by weights, obtaining a nearly optimal ϕ^+ for verification.

In terms of time complexity, GSA has $\mathcal{O}(k^n)$ in the worst and $\mathcal{O}(n)$ in the best cases. This is because GSA does not need to search all possible results. Instead, it provides a list of probable optimal attacks: Each step makes the choice that seems to be the best at the moment, and by combining the results of all steps, we can end up with a solution that is as optimal as possible. Further, each online verification can update information to increase the success rate in subsequent attacks: GSA can dynamically evolve based on new verification results to optimize the search for the real matching.

Attack process. We first show how $\mathcal{A}$ should update the weight matrix w and the matching ϕ^+ based on the online verification result. After implementing GSA, $\mathcal{A}$ needs to online verify u's password $(u,\nu(\phi^+(u)))$ in descending order of $w(u,\phi^+(u))$. Since passwords are Zipf distributed [34], multiple indices can match a same password $\nu(\phi^+(u))$. For notational convenience, we denote $\mathcal{J}_u$ as the set of all columns corresponding to the indices in u's sweetindex list $\mathbf{sil}_u$ that match $\nu(\phi^+(u))$, i.e., $\mathcal{J}_u=\{j|\nu(ID_j)=\nu(\phi^+(u)), ID_j\in\mathbf{sil}_u\}$. At a high level, $\mathcal{A}$ can narrow down a user's candidate indices by updating w based on the columns in $\mathcal{J}_u$ after each verification, ensuring that the user's real index constantly fall into her candidate indices and thus the removal of decoy indices. In more detail, for each user $u_i^*\in\mathcal{U}$, $\mathcal{A}$ is faced with the following two cases:

- **Verification fails.** $\mathcal{A}$ fails to log in to user u_i^*'s account with the password $\nu(\phi^+(u_i^*))$, which means that the index $\phi^+(u_i^*)=ID_{j^*}$ cannot be u_i^*'s real index. In this case, $\mathcal{A}$ updates the weight matrix by setting w_{i^*,j^*}=0, *removing this index and narrowing down u_i^*'s candidate indices.*
- **Verification succeeds.** $\mathcal{A}$ successfully logs in to user u_i^*'s account with the password $\nu(\phi^+(u_i^*))$, which means that u_i^*'s real password is $\nu(\phi^+(u_i^*))$. To update row i^*, $\mathcal{A}$ keeps the $w_{i^*,j}$ for $j\in\mathcal{J}_{u_i^*}$ unchanged and sets $w_{i^*,j}$=0 for all $j\notin\mathcal{J}_{u_i^*}$, *ensuring that u_i^*'s real index is constantly in $\mathcal{J}_{u_i^*}$.* Further, to update the column j^*, $\mathcal{A}$ first checks whether the identified password has only one index ID_j^* in u_i^*'s sweetindex list $\mathbf{sil}_{u_i^*}$ (i.e., $|\mathcal{J}_{u_i^*}|=|\{j^*\}|=1$). If so, $\mathcal{A}$ updates the column j^* by setting w_{i,j^*}=0 for all $i\neq i^*$, as the corresponding index must be u_i^*'s real index. Otherwise, $\mathcal{A}$ will not update the column j^*. *Such operations remove incorrect indices from other users and thus narrow down their candidate indices.*

After these steps, $\mathcal{A}$ executes GSA again to obtain a new matching ϕ^+ and continues the online verification process.

Algorithm 2: Drawing the flatness graph via our attack.

Input : All users' sweetindex lists $\{\mathbf{sil}_u \mid u \in \mathcal{U}\}$.
Output : A vector $\mathcal{V}$ showing the flatness graph.

begin
  $\mathcal{V} \leftarrow \emptyset$, $\mathbf{ckl} \leftarrow \emptyset$; /*$\mathbf{ckl}$ collects cracked user–password pairs */
  for $x \leftarrow 1$ **to** k **do**
    $\phi^+ \leftarrow \mathrm{GSA}(\{\mathbf{sil}_u \mid u \in \mathcal{U}\})$; /*Employ Alg. 1 to find the most likely valid bijection.*/;
    for $u_i \in \mathcal{U}$=$\{u_1, u_2, \cdots, u_n\} \wedge u_i \notin \mathbf{ckl}$ **do**
      $ID_j \leftarrow \phi^+(u_i)$;
      if FAIL $= \mathrm{LOGIN}(u_i, \nu(ID_j))$ **then**
        $w_{i,j} \leftarrow 0$; /* Remove incorrect mapping between u_i and ID_j. */
      else
        $\mathbf{ckl}$.Append((u_i, ID_j));
        $\mathcal{J}_{u_i} \leftarrow \{j' | \nu(ID_{j'}) = \nu(ID_j), ID_{j'} \in \mathbf{sil}_{u_i}\}$;
        for $j' \notin \mathcal{J}_{u_i}$ **do**
          $w_{i,j'} \leftarrow 0$; /* row update */
        if $|\mathcal{J}_{u_i}| = 1$ **then**
          for $i' \neq i$ **do**
            $w_{i',j} \leftarrow 0$; /* column update */
    $N_x \leftarrow |\mathbf{ckl}|$; $\mathcal{V}$.Append($(x, N_x/n)$);
  return $\mathcal{V}$

Algorithm 3: Drawing success-number graph via our attack.

Input : All users' sweetindex lists $\{\mathbf{sil}_u | u \in \mathcal{U}\}$.
Output : A vector $\mathcal{V}$ showing the success-number graph.

begin
  $\mathcal{V} \leftarrow \emptyset$; $\mathbf{ckl} \leftarrow \emptyset$, $\mathbf{fhl}_{1,n} \leftarrow \mathbf{0}$, $x \leftarrow 0$; /* $\mathbf{fhl}$ records a user's number of honeyword logins, $\mathbf{ckl}$ records cracked passwords. */
  while $x < \mathcal{T}_2$ **do**
    $\phi^+ \leftarrow \mathrm{GSA}(\{\mathbf{sil}_u | u \in \mathcal{U}\})$;
    $\mathbf{sul}$=$\{u_i, u_2, \cdots, u_s | u_i \notin \mathbf{ckl}, w(u_i, \phi^+(u_i)) \geq w(u_j, \phi^+(u_j)), \forall i \geq j\}$; /* Sort user list based on $w(u, \phi^+(u))$. */
    for $u_i \in \mathbf{sul}$ **do**
      if $\mathbf{fhl}_{u_i} < \mathcal{T}_1$ **then**
        $ID_j = \phi^+(u_i)$;
        if *fail* $= \mathrm{LOGIN}(u_i, \nu(ID_j))$ **then**
          $w_{i,j} \leftarrow 0$, x++, $\mathbf{fhl}_{u_i}$++;
        else if *success* $= \mathrm{LOGIN}(u_i, pw_j)$ **then**
          $\mathbf{ckl}$.Append($(u_i, \nu(ID_j))$);
          $\mathcal{J}_{u_i} \leftarrow \{j' | \nu(ID_{j'}) = \nu(ID_j), ID_{j'} \in \mathbf{sil}_{u_i}\}$;
          for $j' \notin \mathcal{J}_{u_i}$ **do**
            $w_{i,j'} \leftarrow 0$; /* row update */
          if $|\mathcal{J}_{u_i}| = 1$ **then**
            for $i' \neq i$ **do**
              $w_{i',j} \leftarrow 0$; /* column update */
        $y = |\mathbf{clk}|$, $\mathcal{V}$.Append((x, y));
  return $\mathcal{V}$;

As shown above, updating the weight matrix is required after each online verification, and implementing GSA is likewise required if a new ϕ^+ is to be tried. This is different from prior attacks in [35], [38], where $\mathcal{A}$ straightforwardly online verifies the next sweetword ranked in descending order of Eq. 5. In short, our process reveals how $\mathcal{A}$ can adaptively select the next $(u, \nu(\phi^+(u)))$ to enhance attack efficiency.

For per-user attacks, $\mathcal{A}$'s goal is to identify a particular user's password with as high a success rate as possible before triggering the per-user $\mathcal{T}_1$ (e.g., 3) threshold. Due to correlations among users' sweetwords, $\mathcal{A}$ should traverse all $u \in \mathcal{U}$ to exploit feedback in identifying other users' passwords as much as possible. More concretely, for each user u (not necessarily the target u_i^*), $\mathcal{A}$ should online verify her sweetword $\nu(\phi^+(u))$ given by the obtained matching ϕ^+, and update the weight matrix as well as ϕ^+. The attack stops once $\nu(\phi^+(u_i^*)) = pw_{u_i^*}$. By counting the number of successfully identified passwords (i.e., cracked accounts) under each number of per-user sweetword login attempts, the flatness graph [35] (see Sec. II-C) can be plotted. The process is shown in Alg. 2.

For system-wide attacks, $\mathcal{A}$'s goal is to identify as many passwords as possible before triggering the system-wide $\mathcal{T}_2$ threshold (e.g., 10^4). Hence, once an online verification succeeds, $\mathcal{A}$ should update the weight matrix and move to the next user *given by the order* $w(u, \phi^+(u))$. By counting the number of successful logins under each number of failed honeyword logins, the success-number graph [35] (see Sec. II-C) can be plotted. The process is shown in Alg. 3.

D. Intersection attack theory

We further propose the intersection attack theory for attackers capable of obtaining multiple snapshots of the password file. This is reasonable, since $\mathcal{A}$ may obtain different snapshots before and after a regular update. Note that when all others' indices are treated as a user's decoy indices, e.g., $\mathbf{sil}_u = \mathcal{ID}$ in Superword [15] and Superword+, the sweetindex/sweetword list remains the same if there are no newly registered users. Hence, we mainly consider the case with $|\mathbf{sil}_u| = k < n$.

We first study the case where the number of users is fixed. Formally, we assume that $\mathcal{A}$ can obtain each user u's multiple sweetindex lists $\mathbf{sil}_u^t$, where $t = 1, 2, ..., T$ and T is the number of breached sweetindex list files for each user. In this way, $\mathcal{A}$ can obtain T different sweetindex lists for users who firstly occurred in $\mathcal{A}$'s initial breached, T–1 for users who firstly occurred in $\mathcal{A}$'s second breached, and so forth. This enables $\mathcal{A}$ to intersect and get a narrowed sweetindex list with fewer indices, i.e., $\mathbf{sil}_u^* = \bigcap_{t=1}^{T} \mathbf{sil}_u^t$. The following Eq. 9 gives the probability that the number of decoy indices in a sweetindex list is reduced from k_t to k_{t+1} through an intersection:

$$p(k_t, k_{t+1}) = \frac{\binom{k_t}{k_{t+1}} \cdot \binom{n-1-k_t}{k_t - k_{t+1}}}{\binom{n-1}{k_t}}, \tag{9}$$

where $0 \leq k_t \leq k_{t-1}$ and $p(0, 0) = 1$; In the denominator, $\binom{n-1}{k_t}$ is the number of combinations of k_t decoy indices. In the numerator, $\binom{k_t}{k_{t+1}}$ is the number of combinations that can make k_{t+1} decoy indices remain after an intersection upon a sweetindex list with k_t decoy indices, and $\binom{n-1-k_t}{k_t - k_{t+1}}$ is the number of combinations that can keep these k_{t+1} decoy indices.

Since intersections are mutually independent, the $\mathcal{A}$'s success rate is the product of (1) the summation of all probabilities that reduce the number of decoy indices from k_1 to k_t through $T-1$ intersections, and (2) randomly selecting one as the real password in the narrowed sweetindex list. Formally, there is

$$\Pr[sw = pw_u] = \sum_{\substack{k_1 = k-1 \\ 0 \leq k_{t+1} \leq k_t}} \frac{1}{k_T + 1} \prod_{t=1}^{T-1} p(k_t, k_{t+1}), \tag{10}$$

After intersection attacks, $\mathcal{A}$ can perform an efficient Bayesian attack via GSA in Sec. IV-C on narrowed sweetindex lists $\mathbf{sil}_{u \in \mathcal{U}}^*$ for more efficient distinguishing attacks.

We also consider the case where the user numbers can change. This is realistic, as users can constantly register and delete their accounts. To make it simple, we mainly consider the case with

increasing user numbers, while the case with decreasing user numbers is similar. *We claim that the attacker $\mathcal{A}$ can always identify newly registered users and their associated sweetindex.* Suppose at the end of the time step t, there are n' new users and n' new indices added to the system. Thus, the new users $\Delta\mathcal{U}^{t+1}$=$\{u\in\mathcal{U}^{t+1}\}\backslash\{u\in\mathcal{U}^{t}\}$, where $\mathcal{U}^t$ denotes the set of all users at time t. Similarly, the newly added indices can be obtained by subtracting the union of sweetindex lists at time t+1 with t, i.e., $\Delta\mathbf{sil}_u^{t+1}$=$\{\bigcup_{u\in\mathcal{U}^{t+1}}\mathbf{sil}_u^{t+1}\}\backslash\{\bigcup_{u\in\mathcal{U}^{t}}\mathbf{sil}_u^{t}\}$. In this case, $\mathcal{A}$ can separate the newly added usernames and their sweetindices. As an extreme case, when n'=1, $\mathcal{A}$ can directly identify the new user's real index and the associated password.

Discussion. The above analysis enables us to quantify the narrowing effects of intersection attacks. For instance, when T=2, n=10^4, k_t=n–1 and k_{t+1}=1 in Eq. 9, the probability that no decoy indices remain in the narrowed sweetindex list in two intersections is reduced from 85.84% to 23.85% if k increases from 40 to 120. When $T\geq$3, based on Eq. 10, the probability of directly identifying the real password is over 98%. Therefore, the regular update mechanism seriously backfires when $\mathcal{A}$ can obtain multiple, especially $T\geq$3 snapshots, calling for external mitigations to more responsively detect $\mathcal{A}$'s presence. In this regard, we mainly explore the case with T=1,2 in our experiments, and present detailed results in Sec. V-D.

E. Denial-of-service (DoS) attacks

We also explore whether internally sampled honeyword schemes can resist DoS attacks, i.e., to make the false breach alarm rate as small as possible. In DoS attacks, the attacker $\mathcal{A}$ first registers m (e.g., m=10^4) accounts with the same password pw', and logs in with pw' on the n user accounts to trigger alarms after regular updates [13]. Based on these premises, the probability that a user account triggers false alarms is

$$\Pr[|hw|\geq\mathcal{T}_1]\geq\sum_{j=\mathcal{T}_1}^{k-1}\binom{k-1}{j}Q^j(1-Q)^{k-1-j}, \tag{11}$$

where $|hw|$ is the number of matched honeywords, $\mathcal{T}_1$ is the per-user honeyword login threshold ($\mathcal{T}_1$=3) that triggers the alarm of a user account, and Q=$m/(n$+$m)$ is the probability that pw' is a user's honeyword. The equality holds if pw' has not been used by the n users (e.g., a random password).

Generality of our framework and attack theories. Although there exist only two major sound internally sampled honeyword schemes, i.e., Honeyindex+ and Superword+, our generic framework and attack theories in Secs. III&IV are still important. This is because our theories can be applied to sound internally sampled honeyword schemes that might be proposed in the future. As revealed in Sec. III, different instantiations are mainly due to different chains of indices (from a user's password hash to her account), so new internally sampled schemes can be identified once proposed. Our attack theory also does not rely on particular instantiations, so future internally sampled honeyword schemes are also vulnerable to our Bayesian and intersection attacks.

V. Extensive Evaluations

We first introduce the large-scale password datasets used for evaluations. We then provide our experimental setups, instantiate our attacker models and evaluate results.

TABLE III: Basic info of password datasets for evaluating $\mathcal{A}_1$ attacker.

Dataset	Language	Service type	Leaked time	Original size
4iQ	Mixed	Mixed	Dec. 2017	17,833,311
000Webhost	English	Web hosting	Oct. 2015	10,580,777
Clixsense	English	Paid task platform	Sep. 2016	1,628,018
QNB	English	E-bank	Apr. 2016	75,193
Wishbone	English	Mobile app	Jan. 2020	6,035,833
Taobao	Chinese	E-commerce	Feb. 2016	11,633,762
126	Chinese	Email	Oct. 2015	14,154,759
12306	Chinese	Train ticketing	Dec. 2014	117,808

TABLE IV: Basic info of PII-associated datasets that are constructed by matching emails for evaluating $\mathcal{A}_2$ attacker.

Dataset	Language	Size	Type of PII used
Clixsense	English	118,124	Email
000Webhost	English	130,823	Email, Username, Name, Birthday
12306	Chinese	129,295	Email, Username, Name, Birthday
Rootkit	English	14,903	Email, Username, Name, Birthday

A. Our datasets and ethical considerations

Dataset. We evaluate the attack efficiency of our attackers characterized by different guessing capacities in using auxiliary information (e.g., public datasets, PII, and sister passwords). In our evaluation, we use Rockyou to train our attackers, which was leaked as early as Dec., 2009 and contains 32 million passwords. It can provide a sufficiently large password sample for characterizing early users' password behaviors.

Table III shows eight large-scale password datasets used to evaluate the attacker $\mathcal{A}_1$, including four from English sites, three from Chinese sites, and one (i.e., 4iQ) from mixed sites. These datasets are from various types of services and are widely used in research [38]–[40]. Some early disclosed datasets (e.g., 000Webhost and 12306) and Wishbone is a dataset leaked four years ago with more recent user passwords. Table IV shows four password datasets that contain personally identifiable information (PII) used to evaluate the attacker $\mathcal{A}_2$. Particularly, Clixsense only contains emails, while 000Webhost, 12306 and Rootkit datasets contain emails, birthdays and (user)names. Table V shows five datasets consisting of pairs of sister passwords used to evaluate the attacker $\mathcal{A}_3$. In particular, we removed the same sister passwords from these datasets. This is reasonable because: (1) The same sister passwords would render any honeyword scheme ineffective; (2) There are many protocols (e.g., PCR [42], HIBP [2], and Google Password Checkup [22]) to address the problem of exact password reuse.

TABLE V: Basic info of sister password datasets that are constructed by matching emails for evaluating $\mathcal{A}_3$ attacker.

Dataset	Size	Language
000Webhost-LinkedIn	22,514	English
Clixsense-LinkedIn	102,241	English
Clixsense-Yahoo	15,883	English
Tianya-Dodonew	523,312	Chinese
Tianya-Taobao	368,127	Chinese

Ethical considerations. We only provide aggregated statistical information and treat each account as confidential without revealing the contents. All password datasets are stored and processed on computers that are not connected to the Internet. Once the data analysis is completed, the recovered password hashes are deleted. Hence, the usage of these datasets in our research will not increase the risk to the victims involved. We have also obtained approval from the IRB of our institution.

B. Instantiating our attacker models

We instantiate $\Pr_{\mathrm{PW}}[sw|\mathrm{X}_u]$ in Eqs. 1 and 4 for $\mathcal{A}_1$∼$\mathcal{A}_3$ attackers with different capacities as follows,

$$\Pr_{\mathrm{PW}}[sw \mid \mathrm{X}_u] = \begin{cases} \frac{f_D(sw)+\alpha}{|D|+|\mathbf{swl}_u|} & \mathrm{X}_u{=}\perp \\ \max\limits_{\mathrm{PII}_u \in \mathrm{PII}} \mathsf{sim}(f(sw), f(\mathrm{PII}_u)) & \mathrm{X}_u{=}\mathrm{PII}_u \\ \mathsf{sim}(f(sw), f(pw_u^s)) & \mathrm{X}_u{=}pw_u^s \end{cases}, \quad (12)$$

where D represents the auxiliary dataset (i.e., `Rockyou`) used by attackers, $f_D(sw)$ is the frequency of the sweetword sw in D, and α=1 is the smoothing parameter. To calculate the similarity between a sweetword sw and a user's PII_u or sister password pw_u^s, we employ the state-of-the-art Huang et al.'s method [17], which is comprised of: (1) a transformer-based function $f(\cdot)$:$\{0,1\}^*$→$\mathbb{R}^d$ [32] that takes a string (ranging from sweetword sw to a user's PII_u and pw_u^s) as the input, and outputs latent representations in the embedding space of the dimension d; (2) Chen et al.'s widely-employed cosine similarity metric $\mathsf{sim}(\cdot,\cdot)$: $\mathbb{R}^d$×$\mathbb{R}^d$→$[0,1]$ [7].

C. Experimental setups

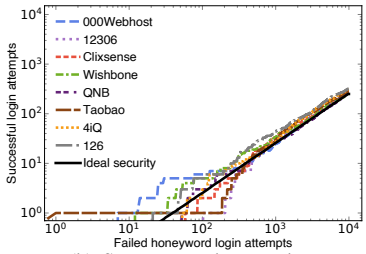
As shown in Sec. III-C, Honeyindex+ and Superword+ are different instantiations under the same generic framework, so we evaluate them with similar settings. This means that once the attacker $\mathcal{A}$'s submitted password is correct, she can always log in (see Sec. III-C). Although honeypots (i.e., fake accounts registered by the server) can be used to detect password leakage, we adopt the more conservative assumption that password leakages are detected via honeyword login attempts.

We mainly set the number of user accounts n=10^4 in our experiments. For Honeyindex+, as in [35], [38], we set per-user threshold $\mathcal{T}_1$=3, the system-wide threshold $\mathcal{T}_2$=10^4, and the number of sweetwords k=40. For Superword+ we use the same $\mathcal{T}_1$ and $\mathcal{T}_2$ settings and set k=n=10^4. When the number of honeyword login attempts for a user reaches $\mathcal{T}_1$, a per-user alarm is triggered; when the total number across all accounts reaches $\mathcal{T}_2$, a system-wide alarm is triggered. Thus, the attacker can attempt at most $\mathcal{T}_1$=3 guesses per account and must stop after $\mathcal{T}_2$=10^4 honeyword logins overall. Unless stated otherwise, we use $(k, \mathcal{T}_1, \mathcal{T}_2)$=$(40, 3, 10^4)$ for Honeyindex+ and $(10^4, 3, 10^4)$ for Superword+. For more comprehensive evaluations, we also vary k∈$\{40, 80, 120\}$, $\mathcal{T}_1$∈$\{1, 3, 5\}$, and $\mathcal{T}_2$∈$\{10^3, 10^4, \infty\}$. Here, $\mathcal{T}_2$=∞ means no system-wide alarm, which can be achieved by setting $\mathcal{T}_2$>$\mathcal{T}_1 \cdot n$.

For DoS attacks, we assume that $\mathcal{A}$ can register m=10^4 additional accounts to trigger alarms. For intersection attack, we conservatively assume that $\mathcal{A}$ can obtain T=2 snapshots of the password file when the number of users is n=2,000 and 10^4. For T≥3, the attacker can identify the real password with probability over 98% (see Sec. IV-D); thus, these cases are omitted. Specifically, we randomly select n=10^4 accounts from each dataset in Tables III–V. For Honeyindex+, we include a baseline representing the ideal $1/k$ security, where attacks reduce to random guessing (see Fig. 3); For Superword+ with k=n, the baseline is omitted as ideal $1/n$ security implies that only one account can be cracked. Flatness and success-number graphs are generated by Algs. 2 and 3, respectively.

Dataset	Wang et al.'s [38]
000Webhost	2.38%
12306	2.69%
Clixsense	2.58%
Wishbone	2.64%
QNB	2.59%
Taobao	3.04%
4iQ	3.15%
126	3.06%
Average	**2.77%**

(a) ϵ-flatness information



(b) Success-number graph

Fig. 2: ϵ-flatness (i.e., the average success rate of online guess once per account) and success-number graph of Honeyindex+ against the attacker $\mathcal{A}_1$ using Wang et al.'s naive Bayesian attack (instantiated by Theorem 2) under the system parameters k=40, $\mathcal{T}_1$=3, and $\mathcal{T}_2$=10^4. The ideal security that reduces distinguishing attacks to random guesses does not hold since ϵ≥2.50%(=$1/40$).

D. Experimental results

Invalidation of ideal security. Fig. 2 presents the results of Honeyindex+ against $\mathcal{A}_1$ using Wang et al.'s naive Bayesian attacks [38], where no correlations between different users' sweetwords are exploited. For per-user attacks, Fig. 2(a) shows that with only one guess per account, the success rate can exceed the ideal 2.50%(=$1/40$) security. This substantiates our analysis in Sec. IV-B that the internally sampled honeyword scheme is hard to achieve the ideal $1/k$ security even against the naive Bayesian attack. For system-wide attacks, as Fig. 2(b) and Table VI show, the naive Bayesian attack [38] identifies 2.53%∼3.29% of real passwords with 10^4 failed honeyword logins, which exceeds the ideal 2.50%(=$1/40$) security.

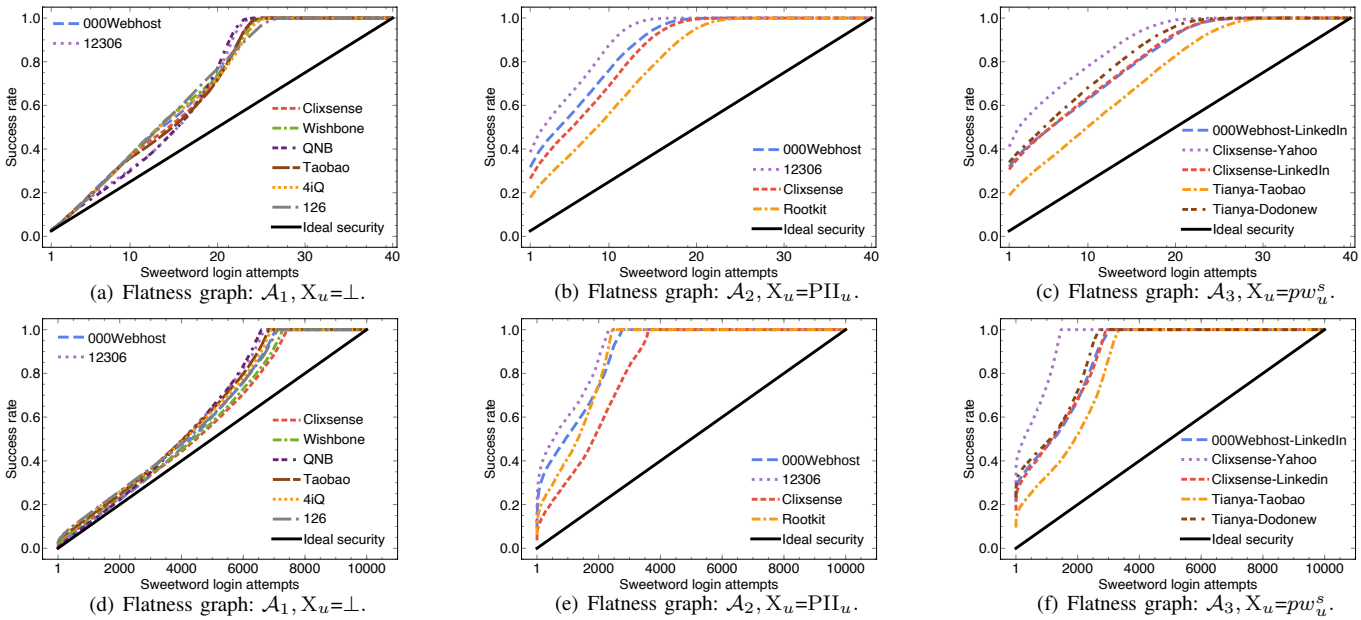
Comparison between our and Wang et al.'s naive attacks [38]. Our Bayesian attack supported by GSA (see Sec. IV-C) is more efficient than the naive attack [38]. For per-user attacks, with only one guess, $\mathcal{A}_1$'s success rate can reach 3.82%∼4.12%, which is 1.26∼1.34 times the naive attack. Although the increased success rate of our attack over the naive one seems small (i.e., 0.79%∼1.05% in the absolute term), the advantage becomes more pronounced as the number of user accounts grows. For instance, the increased success rate scales from 0.79%∼1.05% to 1.46%∼2.09% as n grows from 10^4 to 10^5. *This means that as the number of user accounts grows, both the proportion and the absolute number of identified passwords will increase, making internally sampled honeyword schemes increasingly insecure. Since account compromises can result in catastrophic consequences (e.g., $4.4 million cryptocurrency loss from about 25 victims' accounts [28]), the increased success rate can be threatening in practice.*

For system-wide attacks, within $\mathcal{T}_2$=10^4 failed honeyword logins, $\mathcal{A}_1$ can identify 4.31%∼5.04% of real passwords, which is 1.36∼1.62 times the naive one. These results reveal the effectiveness of our Bayesian attacks, with only three exceptions in $(\mathcal{T}_1, \mathcal{T}_2)$=$(1, 10^3)$. This is explainable, as conflict handling in GSA (see Alg. 1) may make $\mathcal{A}_1$'s initially selected sweetwords less probable. We use an example to explicate this. Suppose the valid bijection ϕ^+ maps both users u_{i_1} and u_{i_2} to their shared sweetindex id_j, i.e., $\phi^+(u_{i_1})$=$\phi^+(u_{i_2})$=id_j, and $\nu(\phi^+(u_{i_2}))$=pw_j is u_{i_2}'s most probable sweetword. Based on the conflict handling in GSA (see Sec. IV-C), u_{i_2}'s sweetindex will be reset to the next available id_{j*}, so pw_j will not be online verified first as in the naive attack [38], leading to lower

TABLE VI: Distinguishing attack success rates (our Bayesian attack achieved by GSA vs. Wang et al.'s naive Bayesian attack [38]) against Honeyindex+ under varied system parameter settings $(k, \mathcal{T}_1, \mathcal{T}_2)$ and n=10^4.

$(k, \mathcal{T}_1, \mathcal{T}_2)^\dagger$	$\mathcal{A}_1, \mathrm{X}_u = \perp$			$\mathcal{A}_2, \mathrm{X}_u$=$\mathrm{PII}_u$			$\mathcal{A}_3, \mathrm{X}_u$=$pw_u^s$		
	Our attack	Wang et al.'s [38]	Increased‡	Our attack	Wang et al.'s [38]	Increased‡	Our attack	Wang et al.'s [38]	Increased‡
$(40, 1, 10^3)$	0.28%∼0.47%	0.20%∼0.39%	0.94∼**1.57**	10.1%∼25.9%	6.79%∼22.7%	**1.11∼1.14**	12.2%∼31.5%	11.2%∼27.3%	**1.04∼1.15**
$(40, 3, 10^4)$	4.31%∼5.04%	2.53%∼3.29%	**1.36∼1.62**	18.6%∼44.8%	14.0%∼36.5%	**1.14∼1.21**	20.6%∼43.6%	19.4%∼35.5%	**1.05∼1.23**
$(40, 1, \infty)$	3.82%∼4.12%	2.38%∼3.15%	**1.26∼1.34**	17.8%∼38.6%	13.0%∼32.1%	**1.12∼1.18**	18.9%∼41.3%	18.2%∼33.3%	**1.03∼1.12**
$(40, 3, \infty)$	10.9%∼12.9%	7.35%∼8.34%	**1.31∼1.58**	26.0%∼51.9%	20.3%∼41.4%	**1.16∼1.24**	27.0%∼54.7%	24.7%∼38.9%	**1.08∼1.16**
$(40, 5, \infty)$	17.4%∼21.2%	12.2%∼13.4%	**1.34∼1.67**	33.9%∼59.9%	25.7%∼46.5%	**1.22∼1.29**	34.5%∼62.7%	30.1%∼42.7%	**1.14∼1.22**
$(80, 1, 10^3)$	0.12%∼0.30%	0.12%∼0.19%	0.93∼**1.84**	9.21%∼22.5%	5.71%∼19.4%	**1.15∼1.34**	10.6%∼30.2%	9.23%∼26.0%	**1.01∼1.16**
$(80, 3, 10^4)$	1.74%∼2.30%	1.11%∼1.41%	**1.23∼1.53**	15.8%∼38.7%	11.3%∼31.6%	**1.12∼1.22**	17.4%∼37.5%	16.8%∼33.1%	**1.03∼1.13**
$(80, 1, \infty)$	1.68%∼2.36%	1.09%∼1.40%	**1.25∼1.51**	14.6%∼33.9%	10.9%∼28.0%	**1.09∼1.18**	16.1%∼36.4%	16.0%∼31.5%	**1.01∼1.10**
$(80, 3, \infty)$	5.02%∼6.19%	3.53%∼4.11%	**1.31∼1.59**	21.8%∼44.1%	15.8%∼36.1%	**1.14∼1.21**	22.1%∼47.0%	20.7%∼35.6%	**1.05∼1.11**
$(80, 5, \infty)$	8.80%∼10.8%	6.12%∼6.60%	**1.34∼1.64**	23.9%∼49.9%	19.2%∼40.6%	**1.15∼1.21**	25.9%∼52.8%	23.9%∼38.2%	**1.09∼1.14**
$(120, 1, 10^3)$	0.09%∼0.24%	0.03%∼0.13%	0.94∼**1.96**	8.46%∼20.7%	5.02%∼17.9%	**1.12∼1.16**	9.62%∼27.3%	8.33%∼25.3%	**1.02∼1.16**
$(120, 3, 10^4)$	0.98%∼1.43%	0.77%∼0.93%	**1.08∼1.57**	14.0%∼35.2%	10.0%∼29.3%	**1.10∼1.19**	16.1%∼37.6%	15.8%∼32.0%	**1.02∼1.18**
$(120, 1, \infty)$	1.11%∼1.46%	0.79%∼0.88%	**1.27∼1.66**	13.6%∼31.5%	9.71%∼26.2%	**1.13∼1.20**	15.4%∼33.0%	15.3%∼30.6%	**1.01∼1.05**
$(120, 3, \infty)$	3.38%∼4.07%	2.47%∼2.67%	**1.30∼1.55**	17.8%∼40.8%	13.6%∼33.3%	**1.12∼1.21**	19.4%∼43.8%	19.0%∼34.4%	**1.02∼1.08**
$(120, 5, \infty)$	6.15%∼7.36%	4.12%∼4.28%	**1.49∼1.75**	20.9%∼45.1%	16.4%∼37.1%	**1.14∼1.21**	22.7%∼48.4%	21.2%∼36.5%	**1.05∼1.11**

† When $\mathcal{T}_2$=∞, no system-wide alarms will be triggered, and each user account allows at most $\mathcal{T}_1$ failed honeyword login attempts.
‡ All % are obtained by dividing 10^4 accounts. The "Increased" column is calculated by dividing the value under Our attack (see Sec. IV-A) by Wang et al.'s [38].



(a) Flatness graph: $\mathcal{A}_1, \mathrm{X}_u$=$\perp$. (b) Flatness graph: $\mathcal{A}_2, \mathrm{X}_u$=$\mathrm{PII}_u$. (c) Flatness graph: $\mathcal{A}_3, \mathrm{X}_u$=$pw_u^s$.

(d) Flatness graph: $\mathcal{A}_1, \mathrm{X}_u$=$\perp$. (e) Flatness graph: $\mathcal{A}_2, \mathrm{X}_u$=$\mathrm{PII}_u$. (f) Flatness graph: $\mathcal{A}_3, \mathrm{X}_u$=$pw_u^s$.

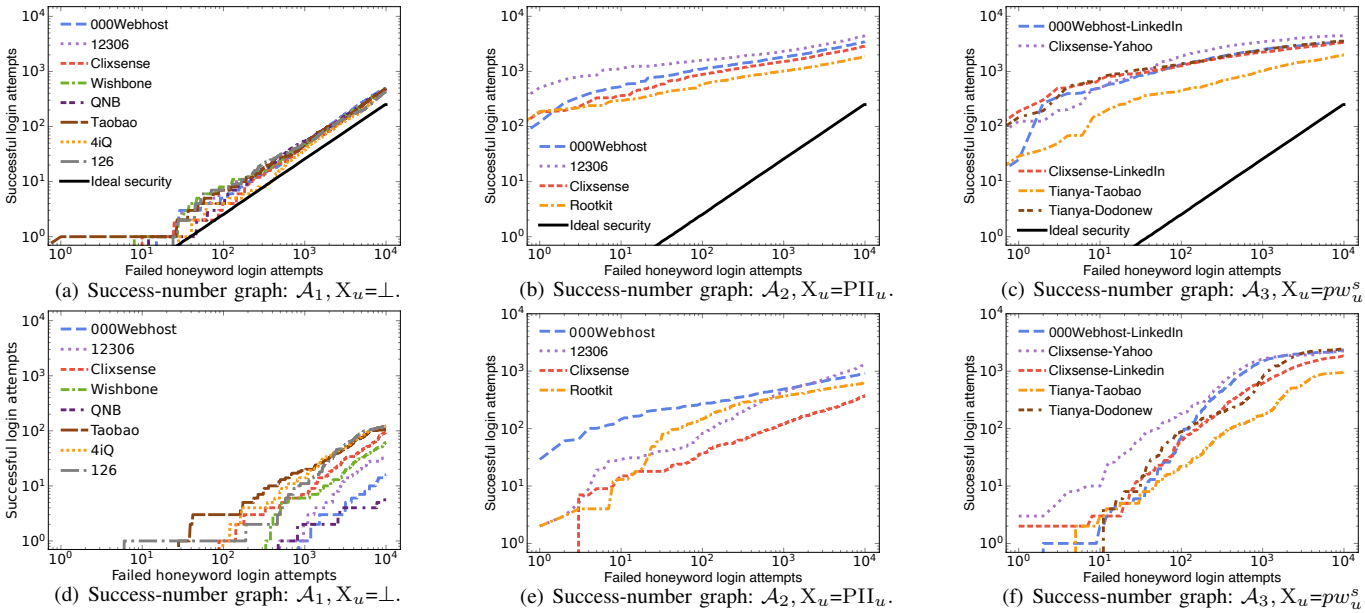
Fig. 3: Flatness graphs measuring $\mathcal{A}_1$∼$\mathcal{A}_3$ per-user distinguishing capability against Honeyindex+ (with k=40) and Superword+ (with k=n=10^4). These schemes are instantiated under our framework (see III-C). All experiments are conducted based on datasets in Tables III∼V, and each graph is plotted by implementing Alg. 2 with different auxiliary information in Eq. 12. The black line represents distinguishing attacks against an ideally secure honeyword scheme: The closer to the black line, the less effective an attack is. The results demonstrate the insecurity of Honeyindex+ and Superword+.

success rates. Nevertheless, as $\mathcal{T}_1$ and $\mathcal{T}_2$ increase, more attack feedback can be exploited to more adaptively choose ϕ^+, and thus facilitates our Bayesian attacks. For example, when $\mathcal{T}_2$=∞ and $\mathcal{T}_1$ increases from 1 to 3, the increased success rate over the naive Bayesian attack of $\mathcal{A}_1$ scales from 0.79%∼1.05% to 2.58%∼4.74% in the absolute term. Further comparisons of $\mathcal{A}_1$∼$\mathcal{A}_3$ with different system parameters are shown in Table VI. Overall, our Bayesian attack is of high attack efficiency.

For $\mathcal{A}_1$ using publicly leaked password datasets. For per-user attacks, Figs. 3(a) and 3(d) show that our Bayesian attack enables $\mathcal{A}_1$ to reach a nearly 100% success rate after about 26 and 7,500 sweetword login attempts when attacking against Honeyindex+ and Superword+, respectively, *without the need to try all sweetwords.* For system-wide attacks, Figs. 4(a) and 4(d) show that our Bayesian attack against Honeyindex+ and Superword+ can make $\mathcal{A}_1$'s success rates 1.72∼2.02 and 6∼123 times the ideal security, respectively. This corroborates our analysis in Sec. IV-B that our Bayesian attack can enhance attack efficiency by choosing valid bijections satisfying $\nu(\phi^+(u))$=pw_u for cracked $(u, \nu(\phi^+(u)))$.

For $\mathcal{A}_2$ using PII and $\mathcal{A}_3$ using sister passwords. For per-user attacks, Figs. 3(b) and 3(c) show that with only *one* online guess against Honeyindex+, our Bayesian attack allows $\mathcal{A}_2$ and $\mathcal{A}_3$ to reach success rates of 17.8%∼38.6% and 18.9%∼41.3%, respectively (see the point $(x$=$1, y)$); Similarly, against Superword+, the one-guess success rates are as high as 3.67%∼12.1% and 9.52%∼29.9% for $\mathcal{A}_2$ and $\mathcal{A}_3$, respectively. For system-wide attacks against Honeyindex+, Figs. 4(b) and 4(c) show that $\mathcal{A}_2$ and $\mathcal{A}_3$ can identify 18.6%∼44.8% and 20.6%∼43.6% of real passwords, respectively, far exceeding the ideal security; Similarly, against Superword+ Figs. 4(e) and 4(f) show that $\mathcal{A}_2$ and $\mathcal{A}_3$ can identify 3.19%∼13.6% and 9.78%∼26.2% of real passwords, respectively.

The above evaluation results against $\mathcal{A}_2$ and $\mathcal{A}_3$ corroborate the inherent limitations of internally sampled honeyword



(a) Success-number graph: $\mathcal{A}_1, \mathrm{X}_u{=}\perp$. (b) Success-number graph: $\mathcal{A}_2, \mathrm{X}_u{=}\mathrm{PII}_u$. (c) Success-number graph: $\mathcal{A}_3, \mathrm{X}_u{=}pw_u^s$.

(d) Success-number graph: $\mathcal{A}_1, \mathrm{X}_u{=}\perp$. (e) Success-number graph: $\mathcal{A}_2, \mathrm{X}_u{=}\mathrm{PII}_u$. (f) Success-number graph: $\mathcal{A}_3, \mathrm{X}_u{=}pw_u^s$.

Fig. 4: Success-number graphs measuring $\mathcal{A}_1{\sim}\mathcal{A}_3$ per-user distinguishing capability against Honeyindex+ and Superword+. The system parameters are set as $(k, \mathcal{T}_1, \mathcal{T}_2){=}(40, 3, 10^4)$. All experiments are conducted based on datasets in Tables III∼V, and each graph is plotted by implementing Alg. 3 with different auxiliary information in Eq. 12. The black line represents distinguishing attacks against an ideally secure honeyword scheme: The closer to the black line, the less effective an attack is. The results demonstrate the insecurity of Honeyindex+ and Superword+ internally sampled honeyword schemes.

TABLE VII: Success number info about Honeyindex+, Superword+, and (Tar)Hybrid [38] under our Bayesian attack instantiated by $\mathcal{A}_1{\sim}\mathcal{A}_3$.[†]

Scheme	$\mathcal{A}_1$, $\mathrm{X}_u{=}\perp$		$\mathcal{A}_2$, $\mathrm{X}_u{=}\mathrm{PII}_u$		$\mathcal{A}_3$, $\mathrm{X}_u{=}pw_u^s$	
	Single PW file	Multiple PW files[‡]	Single PW file	Multiple PW files[‡]	Single PW file	Multiple PW files[‡]
Honeyindex+	4.31%∼5.04%	(18.3%)6.58%∼(18.6%)7.62%	18.6%∼44.8%	(18.3%)21.2%∼(18.5%)45.1%	20.6%∼43.6%	(18.3%)20.7%∼(18.5%)43.7%
Superword+	0.06%∼1.23%	(0.00%)0.12%∼(0.00%)2.48%	3.19%∼13.6%	(0.00%)3.54%∼(0.00%)14.7%	9.78%∼26.2%	(0.00%)9.89%∼(0.00%)26.3%

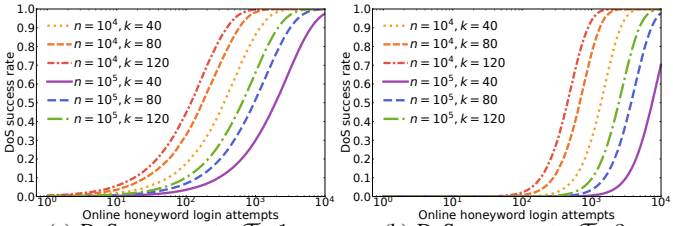
[†] All % are obtained by dividing 10^4 accounts. When $\mathcal{A}_1{\sim}\mathcal{A}_3$ can additionally obtain one PW file when the number of users $n{=}2,000$, they perform the intersection attack (see Sec. IV-D) before the Bayesian attack (see IV-A, which is achieved by the GSA proposed in Sec. IV-C).

[‡] Results in "Multiple files" column are expressed as follows: (the result of the intersection attack)the result of our Bayesian attack after intersection, e.g., (18.2%)6.06%, where 18.2% accounts are identified by the intersection attack and 6.06% additional accounts are identified by our Bayesian attack.

schemes shown in Sec. IV-A, as sampled honeywords are unlikely to be correlated with a user's personal information. As a result, indistinguishability between real passwords and honeywords can be significantly reduced once attackers exploit user-specific personal information. In contrast, externally generated honeyword schemes can mitigate this issue by incorporating user-specific information, and make *generated* honeywords more indistinguishable. For instance, TarHybrid [38], a representative externally generated honeyword scheme, can employ user-specific PII in generating honeywords. Consequently, it reduces $\mathcal{A}_2$'s success rate by 7.20%∼15.1% compared with the internally sampled Honeyindex+.

Effectiveness of our intersection attacks. As shown in Table VII, our intersection attacks alone facilitates $\mathcal{A}_1$ to identify 18.3%∼18.6% of real passwords for Honeyindex+. On this basis, our Bayesian attacks further identify 6.58%∼7.62% of the passwords, i.e., 1.51∼1.53 times compared with a single PW file scenario. This is because the real passwords identified by the intersection attack provide more knowledge, making GSA (see Sec. IV-C) more accurate. For Superword+, since $k{=}n$ at each snapshot, the intersection attack can at best reduce a sweetword list to the smaller size of $k{=}n{=}2{,}000$.

DoS attacks. We conduct DoS attack experiments by varying the number of sweetwords k, user accounts n and per-user honeyword login attempts threshold $\mathcal{T}_1$. Fig. 5 presents the results under $\mathcal{T}_1{=}1$ and 3, and other cases are similar. When $k{=}40$ and $n{=}10^4$, with 1,000 online honeyword login attempts,



(a) DoS success rate: $\mathcal{T}_1{=}1$ (b) DoS success rate: $\mathcal{T}_1{=}3$

Fig. 5: DoS attack success rate against Honeyindex+ when the attacker registers 10^4 user accounts with one unique (e.g., popular) password that has not been used by ordinary users. Each plotted in the log-linear graph means the proportion of a DoS attacker hitting the per-user threshold $\mathcal{T}_1{\in}\{1, 3\}$ with x honeyword login attempts in total (i.e., across all accounts).

DoS success rates are 86.20% and 30.68% for $\mathcal{T}_1{=}1$ and 3, respectively. Using more honeywords mitigates distinguishing attacks, but also significantly increases DoS success rates. For instance, for $\mathcal{T}_1{=}3$ with 10^3 online honeyword login attempts, when k increases from 40 to 80 and 120, the DoS success rates become 2.49 and 3.06 higher, respectively. This calls for additional measures such as rate-limiting policies and CAPTCHA schemes [23], [48] to mitigate DoS attacks.

Discussion. Our major results under $n{=}\mathcal{T}_2{=}10^4$ are reasonable and representative because the attacker $\mathcal{A}$ is unlikely to attempt more than 10^4 accounts. More specifically, $\mathcal{A}$ attempts multiple honeywords per account, so even if $\mathcal{A}$ can make over 10^4 online guesses in total, the number of affected user accounts whose honeywords are logged in is still fewer than 10^4. Our

evaluations also validate this, with the number of affected users for $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ being 6,813∼7,784, 6,193∼8,419, and 6,398∼8,156, respectively. Hence, our evaluations are comprehensive, and reveal the inherent insecurity of internally sampled schemes.

False authentication denials triggered by correct passwords in Honeyindex [13]. As shown in Sec. II-A, Honeyindex only verifies one sweetindex of matched sweetwords. Since duplicate honeywords matching a user's password can be sampled, a user may be unable to log in even with her correct password, causing false authentication denials. To quantify their impacts, consider there are n users, and the frequency of each password is f_{pw}. Hence, the probability of a user u triggering a false denial is $1-\binom{n-f_{pw}}{k-1}/\binom{n-1}{k-1}$, where a f_{pw} ranked r can be estimated by the CDF-Zipf distribution [34], i.e., $f_{pw}{\approx}n{\cdot}C{\cdot}s{\cdot}r^{s-1}$ with $\sum_{pw\in\mathcal{PW}} f_{pw}{=}n$. Accordingly, the expected number of users affected by false denials is $\sum_{u\in\mathcal{U}}(1-\binom{n-f_{pw}}{k-1}/\binom{n-1}{k-1})$, which increases with n and k as shown in Fig. 6. Although the affected fraction is relatively small (0.04%–0.15%), these users will be deterministically denied login, and the absolute number becomes noticeable as n grows. Moreover, once exceeding the system threshold $\mathcal{T}_2$, Honeyindex [13] may falsely trigger a system-wide alarm, making breach detection unreliable.

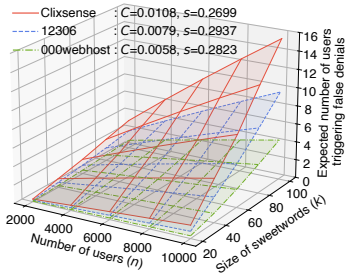


Fig. 6: Expected number of users triggering false denials as a function of the password distribution (C, s), the number of users n and the size of sweetwords k. Accounts that contain popular passwords are more likely to trigger false alarms.

Besides, even if only one sweetindex is verified in a login (see Sec. II-A), Honeyindex [13] offers negligible security gains. For instance, under $(k, \mathcal{T}_1, \mathcal{T}_2){=}(40, 3, 10^4)$, the success rates of $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ (using our Bayesian attack) are still as high as 4.13%∼4.82%, 18.3%∼44.6%, and 20.4%∼43.5%, which are only 4.18%∼4.37%, 0.45%∼1.61% and 0.23%∼0.97% smaller, respectively. In summary, Honeyindex [13] is insecure.

VI. Conclusion

In this work, we have provided a generic framework for sound internally sampled honeyword schemes, and revealed that this technical route is inherently insecure against honeyword distinguishing attacks. We reveal four critical design flaws (two for security and two for usability) in existing Honeyindex [13] and Superword [15] schemes, and for the first time, demonstrate how their variants, Honeyindex+ and Superword+, can be instantiated under our framework. Then, we propose the Bayesian and intersection attack theories to characterize an attacker's optimal distinguishing strategy and propose an efficient algorithm to implement our theories. Guided by our attack theories, we instantiate three attacker models, with each based on different types of available information, against Honeyindex+ and Superword+. Our extensive evaluations on real-world password datasets reveal the inherent flaws of internally sampled honeyword schemes, and neither Honeyindex+ nor Superword+ is secure. We believe this work contributes to advancing the rigorous security analysis of honeyword schemes.

References

[1] *Simpler, Stronger Authentication: Solving the World's Password Problem*, July 2023, https://fidoalliance.org/.
[2] *Have I been pwned: Check if your email address is in a data breach*, 2025, https://haveibeenpwned.com/PwnedWebsites.
[3] Akshima, D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya, "Generation of secure and reliable honeywords, preventing false detection," *IEEE Trans. Depend. Secur. Comput.*, vol. 16, no. 5, pp. 757–769, 2019.
[4] J. Bonneau, C. Herley, P. van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Commun. ACM*, vol. 58, no. 7, pp. 78–87, 2015.
[5] J. Bonneau, C. Herley, P. C. Van 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.
[6] A. Chaabane, G. Acs, M. A. Kaafar *et al.*, "You are what you like! information leakage through users' interests," in *Proc. NDSS 2012*.
[7] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. ICML 2020*, pp. 1597–1607.
[8] S. Croley, *RTX 4090 v6.2.6. Benchmark*, Oct. 2022, https://gist.github.com/Chick3nman/32e662a5bb63bc4f51b847bb422222fd.
[9] A. Dionysiou and E. Athanasopoulos, "Lethe: Practical data breach detection with zero persistent secret state," in *Proc. IEEE EuroS&P 2022*, pp. 223–235.
[10] A. Dionysiou, V. Vassiliades, and E. Athanasopoulos, "Honeygen: Generating honeywords using representation learning," in *Proc. ACM ASIACCS 2021*, pp. 265–279.
[11] M. Dürmuth and T. Kranz, "On password guessing with GPUs and FPGAs," in *Proc. PASSWORDS 2014*, pp. 19–38.
[12] B. Eitel, "Will passwords become a thing of the past?" Mar. 2023, https://www.idsalliance.org/will-passwords-become-a-thing-of-the-past/.
[13] I. Erguler, "Achieving flatness: Selecting the honeywords from existing user passwords," *IEEE Trans. Depend. Secur. Comput.*, vol. 13, no. 2, pp. 284–295, 2016.
[14] X. Gao, Y. Yang, C. Liu, C. Mitropoulos, J. Lindqvist, and A. Oulasvirta, "Forgetting of passwords: Ecological theory and data," in *Proc. USENIX SEC 2018*, pp. 221–238.
[15] Y. Guo, Z. Zhang, and Y. Guo, "Superword: A honeyword system for achieving higher security goals," *Comput. Secur.*, vol. 103, p. 101689, 2021.
[16] Z. Huang, D. Wang, and Y. Zou, "Prob-hashcat: Accelerating probabilistic password guessing with hashcat by hundreds of times," in *Proc. RAID 2024*, pp. 674–692.
[17] Z. Huang, L. Bauer, and M. K. Reiter, "The impact of exposed passwords on honeyword efficacy," in *Proc. USENIX SEC 2024*, pp. 559–576.
[18] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proc. ACM CCS 2013*, pp. 145–160.
[19] M. Kan, "Yahoo execs botched its response to 2014 breach, investigation finds," Mar. 2017, https://www.csoonline.com/article/560433/yahoo-execs-botched-its-response-to-2014-breach-investigation-finds.html.
[20] S. Katsumata, T. Matsuda, W. Nakamura, K. Ohara, and K. Takahashi, "Revisiting fuzzy signatures: Towards a more risk-free cryptographic authentication system based on biometrics," in *Proc. ACM CCS 2021*, pp. 2046–2065.
[21] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
[22] A. D. Kurt Thomas, "Protecting your data, no matter where you go on the web," Feb. 2019, https://blog.google/technology/safety-security/google-password-checkup-cross-account-protection/.
[23] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin, "A measurement study of authentication rate-limiting mechanisms of modern websites," in *Proc. ACSAC 2018*, pp. 89–100.
[24] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE S&P 2014*, 2014, pp. 689–704.
[25] F. Mathis, H. I. Fawaz, and M. Khamis, "Knowledge-driven biometric authentication in virtual reality," in *Proc. ACM CHI 2020*, pp. 1–10.
[26] P. Peng, C. Xu, L. Quinn, H. Hu, B. Viswanath, and G. Wang, "What happens after you leak your password: Understanding credential sharing on phishing sites," in *Proc. ACM ASIACCS 2019*, pp. 181–192.
[27] H. Robert, *Yahoo Raises Breach Estimate to Full 3 Billion Accounts*, Oct. 2017, http://fortune.com/2017/10/03/yahoo-breach-mail/.

[28] H. Shittu, *LastPass Data Breach Results in $4.4 Million Crypto Loss for 25 Victims in a Single Day*, Oct. 2023, https://cryptonews.com/news/lastpass-data-breach-results-in-4-4-million-crypto-loss-for-25-victims-in-a-single-day/.
[29] J. Siegrist, *LastPass hacked - Identified early and resolved*, June 2015, https://blog.lastpass.com/2015/06/lastpass-security-notice/.
[30] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki *et al.*, "Data breaches, phishing, or malware? Understanding the risks of stolen credentials," in *Proc. ACM CCS 2017*, pp. 1421–1434.
[31] K. Toubba, *Notice of Recent Security Incident*, Dec. 2022, https://blog.lastpass.com/posts/2022/12/notice-of-recent-security-incident.
[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NIPS 2017*, p. 6000–6010.
[33] K. C. Wang and M. K. Reiter, "How to end password reuse on the web," in *Proc. NDSS 2019*.
[34] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Foren. Secur.*, vol. 12, no. 11, pp. 2776–2791, 2017.
[35] D. Wang, H. Cheng, P. Wang, J. Yan, and X. Huang, "A security analysis of honeywords," in *Proc. NDSS 2018*, pp. 1–16.
[36] D. Wang, P. Wang, D. He, and Y. Tian, "Birthday, name and bifacial-security: Understanding passwords of Chinese web users," in *Proc. USENIX SEC 2019*, pp. 1537–1555.
[37] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM CCS 2016*.
[38] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang, "How to attack and generate honeywords," in *Proc. IEEE S&P 2022*, pp. 966–983.
[39] D. Wang, Y. Zou, X. Yuan-An, and M. siqi, "Pass2Edit: A Multi-Step generative model for guessing edited passwords," in *Proc. USENIX SEC 2023*, pp. 983–1000.
[40] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. USENIX SEC 2023*, pp. 965–982.
[41] K. C. Wang and M. K. Reiter, "Bernoulli honeywords," in *Proc. NDSS 2024*, pp. 1–18.
[42] K. C. Wang and M. K. Reiter, "Using AMNESIA to detect credential database breaches," in *Proc. USENIX SEC 2021*, pp. 839–855.
[43] 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.
[44] L. Whitney, *LastPass CEO reveals details on security breach*, May 2011, https://www.cnet.com/news/privacy/lastpass-ceo-reveals-details-on-security-breach/.
[45] F. Yu and M. V. Martin, "Honey, I chunked the passwords: Generating semantic honeywords resistant to targeted attacks using pre-trained language models," in *Proc. ICDIMVA 2023*, pp. 89–108.
[46] K. Zetter, *Report: Half a Million Yahoo User Accounts Exposed in Breach*, July 2012, https://www.wired.com/2012/07/yahoo-breach/.
[47] Y. Zhang, F. Monrose, and M. K. Reiter, "The security of modern password expiration: An algorithmic framework and empirical analysis," in *Proc. ACM CCS 2010*, pp. 176–186.
[48] B. B. Zhu, J. Yan, G. Bao, M. Yang, and N. Xu, "Captcha as graphical passwords—a new security primitive based on hard ai problems," *IEEE Trans. Inform. Foren. Secur.*, vol. 9, no. 6, pp. 891–904, 2014.

APPENDIX

Theorem 1. Let $\phi^*\in\Phi^+$ as the user's real bijection, and $\{\mathbf{swl}_u|u\in\mathcal{U}\}$ denote all users' sweetindex lists. Once the authentication server $\mathcal{S}$ is compromised and the F_1 and F_2 files are available, the attacker $\mathcal{A}$ should try in descending order of the following probability

$$\Pr_{\Phi^+}[\phi^*|\{\mathbf{swl}_u|u\in\mathcal{U}\}]=\frac{\Pr_{\Phi^+}[\phi^*]}{\sum_{\phi^+\in\Phi^+}\Pr_{\Phi^+}[\phi^+]}, \tag{13}$$

under the assumption that $\{\mathbf{swl}_u|u\in\mathcal{U}\}$ are independently sampled and a user's real password depends on $\phi^*\in\Phi^+$.

Proof 1: Since different users' sweetindex lists are independently sampled, so are their associated sweetword lists. Hence, for any $\phi^+\in\Phi^+$, we have $\Pr[\{\mathbf{swl}_u,u\in\mathcal{U}\}|\phi^+]=\Pr[\{\mathbf{sil}_u,u\in\mathcal{U}\}|\phi^+]=\prod_{u\in\mathcal{U}}\Pr[\mathbf{sil}_u|\phi^+]$, and $\Pr[\mathbf{sil}_u|\phi^+]\equiv c\ \forall u\in\mathcal{U}$. By Bayesian theorem, we can derive

$$\begin{aligned}&\Pr_{\Phi^+}[\phi^*\mid\{\mathbf{swl}_u,\,u\in\mathcal{U}\}]=\Pr_{\Phi^+}[\phi^*\mid\{\mathbf{sil}_u,\,u\in\mathcal{U}\}]\\&=\frac{\Pr_{\Phi^+}[\phi^*]\Pr[\{\mathbf{sil}_u,\,u\in\mathcal{U}\}\mid\phi^*]}{\sum_{\phi^+\in\Phi^+}\Pr_{\Phi^+}[\phi^+]\Pr[\{\mathbf{sil}_u,\,u\in\mathcal{U}\}\mid\phi^+]}\\&=\frac{\Pr_{\Phi^+}[\phi^*]\prod_{u\in\mathcal{U}}\Pr[\mathbf{sil}_u\mid\phi^*]}{\sum_{\phi^+\in\Phi^+}\Pr_{\Phi^+}[\phi^+]\prod_{u\in\mathcal{U}}\Pr[\mathbf{sil}_u\mid\phi^+]}\\&=\frac{\Pr_{\Phi^+}[\phi^+]}{\sum_{\phi^+\in\Phi^+}\Pr_{\Phi^+}[\phi^+]}.\end{aligned}$$

Ding Wang received his Ph.D. degree in Information Security at Peking University in 2017, and was supported by the "Boya Postdoctoral Fellowship" in Peking University from 2017 to 2019. Currently, he is a Full Professor at Nankai University. As the first author (or corresponding author), he has published over 100 papers at venues like IEEE S&P, ACM CCS, NDSS, USENIX Security, IEEE TIFS and IEEE TDSC. His research has been reported by over 200 media like The Wall Street Journal, Daily Mail, Forbes, IEEE Spectrum, and Communications of the ACM, and resulted in the revision of the guideline NIST SP800-63-2. He has been involved in the community as a PC Chair/TPC member for over 60 international conferences such as NDSS 2023-2025, ACM CCS 2022, PETS 2022-2024, ACSAC 2020-2024, RAID 2024/2023, IEEE EuroS&P 2025, FC 2026/2025, ACM AsiaCCS 2025/2022/2021, ICICS 2018-2025, SPNCE 2020-2022. He has received the "ACM China Outstanding Doctoral Dissertation Award", the Best Paper Award at INSCRYPT 2018, the First Prize of Cryptologic Innovation Award of the China Association for Cryptologic Research, and the First Prize of Natural Science Award of Ministry of Education. His research interests focus on passwords, authentication, and provable security.

Tingwei Fan received the BS degree in cryptography science and technology from the College of Cryptology and Cyber science, Nankai University, Tianjin, P. R. China, in Jun. 2024. He is currently working towards the master degree from the College of Cryptology and Cyber science, Nankai University, Tianjin, P. R. China. He has published papers on ACM CCS, ICICS. His research interests include applied cryptography and password-based authentication.

Fei Duan received the BS degree from the College of Cryptology and Cyber Science, Nankai University, Tianjin, P. R. China, in 2021, and the MS degree in Cryptology and Cyber Science, Nankai University, Tianjin, P. R. China in 2024. He has published papers on IEEE S&P, ACM CCS, IEEE TIFS. His research interests include password security, authentication, and machine learning applications.

Zhenduo Hou received his PhD degree in applied mathematics from Peking University, Beijing, P. R. China in Dec., 2022, and B.S. degree in mathematics from Sichuan University, Chengdu, P. R. China, in June. 2015. Currently, he is a Research Associate of College of Cryptology and Cyber Science of Nankai University, P. R. China. He has published papers on ACM CCS, Science China (information sciences), IEEE TIFS, Inscrypt, etc. He has received the Honorary Mention Recognization of Inscrypt 2024. His research interests include applied cryptography and password-based authentication.