DALock: **Password Distribution-Aware Throttling**

Anonymous Author(s)

ABSTRACT

Large-scale online password guessing attacks are widespread and continuously qualified as one of the top cyber-security risks. The common method for mitigating the risk of online cracking is to lock out the user after a fixed number (K) of consecutive incorrect login attempts. Selecting the value of K induces a classic security-usability trade-off. When K is too large, a hacker can (quickly) break into a significant fraction of user accounts, but when K is too low, we will start to annoy honest users by locking them out after a few mistakes. Motivated by the observation that honest user mistakes typically look quite different from an online attacker's password guesses, we introduce DALock, a *distribution-aware* password lockout mechanism to reduce user annoyance while minimizing user risk. As the name suggests, DALock is designed to be aware of the frequency and popularity of the password used for login attacks. At the same time, standard throttling mechanisms (e.g., K-strikes) are oblivious to the password distribution. In particular, DALock maintains an extra "hit count" in addition to "strike count" for each user, which is based on (estimates of) the cumulative probability of *all* login attempts for that particular account. We empirically evaluate DALock with an extensive battery of simulations using real-world password datasets. In comparison with the traditional K-strikes mechanism, we find that DALock offers a superior security/usability trade-off. For example, in one of our simulations, we are able to reduce the success rate of an attacker to 0.05% (compared to 1% for the 3-strikes mechanism) whilst simultaneously reducing the unwanted lockout rate for accounts that are not under attack to just 0.08% (compared to 4% for the 3-strikes mechanism).

CCS CONCEPTS

• **Security and privacy** → **Usability in security and privacy**; **Authentication**.

KEYWORDS

Authentication Throttling; Password; Dictionary Attack

ACM Reference Format:
Anonymous Author(s). 2022. DALock: Password Distribution-Aware Throttling. In *AsiaCCS '22: ACM ASIA Conference on Computer and Communications Security, May 30– June -3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

AsiaCCS '22, May 30– June -3, 2022, Nagasaki, Japan
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

An online password attacker repeatedly attempts to login to an authentication server submitting a different guess for the target user's password on each attempt. The human tendency to pick weak ("low-entropy") passwords has been well documented, e.g., [5]. An untargeted online attacker will typically submit the most popular password choices consistent with the password requirements (e.g., "Password1"). In contrast, a targeted attacker [44] might additionally incorporate background knowledge about the specific target user (e.g., birthdate, phone number, anniversary, etc.). To protect users against online attackers, most authentication servers incorporate some form of throttling mechanism. In particular, the K-strikes mechanism temporarily locks a user's account if K-consecutive incorrect passwords are attempted within a predefined time (e.g., 24 hours). Setting the lockout parameter K induces a classic security-usability trade-off. Selecting small values of K (e.g., $K = 3$) provides better protection against online attackers but may result in many unwanted lockouts when an honest user miss-types (or miss-remembers) their password. Selecting a larger value of K (e.g., $K = 10$) will reduce the unwanted lockout rate but may increase vulnerability to online attacks.

Bonneau et al. [6] considered many proposed replacements for password authentication, finding that all proposals have some drawbacks compared with passwords. For example, passwords are easier to revoke than biometrics. Similarly, hardware tokens are expensive and require users to carry them around. By contrast, passwords are easy to deploy and do not require users to carry anything around. Put simply, we have not found a "silver bullet" replacement for passwords. Thus, despite all of their shortcomings (and many attempts to replace them), passwords will likely remain entrenched as the dominant form of authentication on the internet [22]. Thus, protecting passwords against online attacks without locking out legitimate users remains a crucial challenge for the foreseeable future [37].

One approach to protect users against online guessing attacks is to adopt strict password composition policies to prevent users from selecting weak passwords. However, it has been well documented that users dislike restrictive policies and often respond in predictable ways [24]. Another defense is to store cookies on the user's device to prove that the next login attempt comes from a known device. Similarly, one can also utilize features such as IP address, geographical location, device, and time of day [17, 32] to help distinguish between malicious and benign login attempts. While these features can be helpful indicators, they are not failproof. Honest users oftentimes travel and login from different devices at unusual times. Similarly, an attacker may attempt to mimic the login patterns of legitimate users. The online attacker can also submit guesses from a wide variety of IP addresses and geographical locations, e.g., using a botnet.

Contributions

We introduce DALock, a novel Distribution-Aware throttling mechanism that can achieve a better balance between usability and security. The key intuition behind DALock is to base lockout decisions on the *popularity* of the passwords that are being guessed. An online attacker will typically want to attempt the most popular passwords to maximize their chances of success. By contrast, when an honest user miss-types (or miss-remembers) their password, the attempt is less likely to be a globally popular password. In addition to keeping track of K_u (the number of consecutive incorrect login attempts), DALock keeps track of a "hit count" Ψ_u for each user u, where Ψ_u intuitively represents the cumulative probability mass of all incorrect login attempts for user u's account. When Ψ_u exceeds the threshold Ψ, we decide to lock the account.

Example 1: Usability **Figure** 1 compares the usability of DALock with the standard 3-strikes mechanism. In this example scenario, our user John Smith registers an account with the somewhat complicated password "J.S.UsesStr0ngpwd!" based on the story "John Smith uses a strong password.". Later, when John tries to login into his account, John remembers the basic story, but not the exact password. Did he use his first name and his last name? With or without abbreviation? Did he add a punctuation mark at the end? Which letters are capitalized? If we use the 3-strikes mechanism, John Smith will be locked out quickly, e.g., after trying the incorrect password guesses "JohnUseStrongPassword," "JohnUsesStrongPassword," and "JohnUsesStrongpwd." However, since none of these passwords is overly popular DALock would allow our user to continue attempting to login until he recovers the correct password.

Example 2: Security **Figure** 1 compares DALock with the 10-strikes mechanism. In this scenario, our user registers an account with a weak password "letmein." Because the password is globally popular, it is likely that an online attacker will attempt this password within the first 10 guesses and break into the account. By contrast, DALock will quickly lock down the account after the attacker submits two globally popular passwords.

To deploy DALock, we need a *frequency oracle* to estimate the frequency of each incorrect login attempt to update Ψ_u. We propose two implementations: password strength models (e.g., ZXCVBN [46]) and a differentially private count sketch data structure. Of course, no frequency oracle will perfectly estimate the true strength of a password and the attacker may try to exploit passwords that are over/underestimated by frequency oracle. We introduce the password knapsack problem to model the optimal (untargeted) attack against DALock. Intuitively, the attacker will try to find a subset of passwords to check which maximizes his success rate subject to the constraint that the total estimated hit count does not exceed the threshold Ψ_u. While password knapsack is NP-Hard, we show that a simple heuristic algorithm works well on empirical datasets.

We then evaluate DALock empirically by simulating an authentication server in the presence of an online password attacker comparing DALock with the traditional K-strikes mechanism for $K \in \{3, 10\}$. In our simulations, we use the password knapsack problem to model the behavior of the attacker and we model honest user login attempts/mistakes using a simple model based on prior empirical studies of password typos [11, 12]. Our experiments show that when the hit count threshold Ψ is tuned appropriately, DALock significantly outperforms K-strikes mechanisms. In particular, when user accounts are under attack, we find that the fraction of accounts that are compromised is significantly lower for DALock than classic K-strikes mechanisms — even for the strict K=3 strikes policy. We also evaluate the unwanted lockout rate of user accounts that are not under attack. We find that the unwanted lockout rate for DALock is much lower compared to K=3 strikes mechanism. The unwanted lockout rate for DALock and the more lenient K=10 strikes mechanism were comparable. We also evaluate the performance of DALock when the organization bans the top B most popular passwords to encourage users to select stronger passwords. We find that DALock continues to outperform the traditional K=3 strikes mechanism in terms of both usability and security. A more detailed description of our experiments can be found in **section** 6.

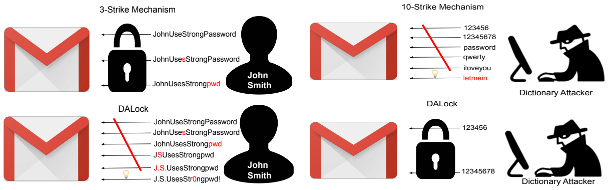

Figure 1: Security(L) & Usabilty(R) Comparison

2 RELATED WORK AND BACKGROUND

2.1 Authentication Throttling

K-strikes Mechanism K-strikes mechanism is a straight-forward implementation for authentication throttling. As its name suggests, throttling occurs when K consecutive incorrect login attempts are detected. To reduce the cost of expensive overhead caused by unwanted throttling, Brostoff [7] et al. suggest setting threshold K to be 10 instead of 3. They argue the increment risk is limited when a strong password policy is enforced. However, this argument is challenged by empirical analyses of password composition policies [4, 24]. Many password composition policies do not rule out all low entropy password choices. For instance, it turns out that banning dictionary words does not increase entropy as expected [24].

Feature-Based Mechanism Modern throttling mechanisms [21, 32] often use features such as geographical location, IP-address, device information, etc., to detect unusual activities. These features can be used to train sophisticated machine learning models to help distinguish between malicious and benign login attempts [17]. DALock takes an orthogonal approach and relies instead on the popularity of the password guesses. One can combine those models with a rigorous throttling system for better performance.

Password Distribution-Aware Throttling In an independent line of work, Tian et al. [34] developed an IP-based throttling mechanism that exploits differences between the distribution of honest login attempts and malicious guesses. In particular, they propose to "silently block" login attempts from a particular IP address ip if the system detects too many popular passwords being submitted from ip. In more detail, StopGuessing uses a data structure called the binomial ladder filter [35] to (approximately) track the frequency $F(pw)$ of each incorrect password guess pw. For each IP address ip, the StopGuessing protocol maintains an associated counter $I_{ip} = \sum_{pw \in \mathcal{P}} F(pw)$ where $\mathcal{P}$ is a list of incorrect password guesses that have been (recently) submitted from ip — I_{ip} can be updated without storing $\mathcal{P}$ explicitly. Intuitively (and oversimplifying a bit) if I_{ip} exceeds a predefined threshold T, then login attempts from address *ip* are silently blocked, i.e., even if the attacker (or honest user) submits a correct password, the system will respond that authentication fails. The authors also suggest protecting accounts with weak passwords by setting a user-specific threshold $T(F(pw_u))$ based on the strength $F(pw_u)$ of the password pw_u of user u. Now, if $I_{ip} > T(F(pw_u))$, the system will silently reject any password from address *ip*. Both StopGuessing and DALock exploit differences between the distribution of user passwords and attacker guesses. One of the key differences is that StopGuessing focuses on identifying malicious IP addresses (by maintaining a score I_{ip} for each IP address *ip*) while DALock focuses on protecting individual accounts by maintaining a "hit-count" parameter Ψu for each user u. There are several other key differences between the two approaches as well. First, in DALock, the goal of our frequency oracle (e.g., count sketch, password strength meter) is to estimate the *total fraction* of users who have actually selected that particular password — as opposed to estimating the frequency with which that password has been *recently* submitted as an incorrect guess. Second, DALock does not require silent blocking of login attempts, which could create usability concerns if an honest user is silently blocked when they enter the correct password.

2.2 Passwords

Password Distribution To analyze online statistical guessing attacks it is important understanding the distribution of user passwords. Password distributions have been extensively studied since the last decades [16, 26]. Using leaked password corpora [5, 31] is a straightforward way to describe the distribution of passwords. Recent works of Wang et al. [41–43] show that password distributions follow Zipf's law, i.e., leaked password corpora nicely fit Zipf's law distributions. Blocki et al. [3] later found that Zipf's law nicely fits the Yahoo! password frequency corpus [2, 5].

Password Typos To test the usability of DALock, it is crucial to reasonably simulate users' mistakes. Recent studies [11, 12] from Chatterjee et al. have summarized probabilities of making (various types of) typos when one enters his or her password based on users' studies. Based on the empirically measured data, they proposed two typo-tolerant authentications without sacrificing security. In fact, similar mechanisms have already been deployed in the industry [18].

2.3 Eliminating Dictionary Attacks

Increasing Cost of Authentication Pinkas and Sanders [29] proposed using puzzles (e.g., CAPTCHAs) as a way to stop online password crackers. CAPTCHAs are hard AI challenges meant to distinguish people from bots [39]. For example, reCAPTCHA [40] has been widely deployed, e.g., Google, Facebook, Twitter, CNN, etc. If we assume that CAPTCHAs are only solvable by people, it is possible to mitigate automated online attacks without freezing users' accounts [8, 9]. Nevertheless, an attacker can always pay humans to solve CAPTCHA challenges [30]. Besides, sophisticated CAPTCHA solvers [47] powered by neural networks make it increasingly challenging to design CAPTCHA puzzles that are also easy for a human to solve. Golla et al. [19] proposed a fee-based password verification system where a small deposit is necessary to authenticate, which is refunded after successful authentication. A password cracker risks losing its deposit if it is not able to guess the real password.

Eliminating Popular Passwords One mediation for dictionary attacks is eliminating the existence of weak or popular passwords. Password composition policy is a common approach, but efforts to force users to pick strong passwords by requiring users to include numbers, capital letters, and/or special symbols have shown limited success [4, 24]. An alternate approach of Schechter et al. [33] is to ban passwords if and only if too many users have picked them using a count-sketch data structure for frequency estimation. A theoretical model by Blocki et al. [4] shows that this is the optimal approach to boost the minimum entropy of the password distribution.

3 PRELIMINARIES

3.1 Count Sketch

The count sketch [10] data structure and its variants are widely used in the tasks for finding frequent items such as popular passwords [28], homepage settings [15]. StopGuessing[34] introduced a new data-structure called a binomial ladder to estimate password frequency, but the frequency estimates can be biased to overestimate the frequency of recently popular passwords. We elect to use the count (median) sketch [10] data structure in this work as it is invariant to the order in which passwords are added. The state $\sigma : \mathsf{R}^{d \times w} \times \mathsf{R}$ of a count sketch (CS) is represented by a two-dimensional $d \times w$ array CS.ARRAY and a total frequency counter CS.T. The data strucutre additionally uses $d + 1$ hash functions $(h_1, \ldots, h_d, h_\pm)$ with the first d functions chosen uniformly at random from a pairwise-independent family:

$$h_1, \cdots, h_d : \{pw\} \to \{1 \cdots w\}, \text{ and } h_\pm : \{pw\} \to \{1, -1\}$$

In this work, we consider the following four classic count sketch APIs: Initialize, Add, Estimate, and TotalFreq. Additionally, we consider an extra operation DP, which is used to construct a differentially private count sketch from a standard one.

$\sigma_0 \leftarrow$ **Initialize(d, w)** : This API initializes and returns a count sketch of state $0^{d \times w} \times 0$, i.e., an all-zero table.

$\sigma_{new} \leftarrow$ **Add(pw, σ):** Intuitively, the Add operation returns an updated count sketch state σ_{new} in which the frequency count of password pw increases by 1.

Given a multiset $\mathcal{D}_\mathcal{U} = \{pw_1, \cdots, pwd_N\}$, we use the following notation $\sigma_{\mathcal{D}_\mathcal{U}} = Add(\mathcal{D}_\mathcal{U}, \sigma)$
$= Add(pw_1, Add(pw_2, Add(pw_3, \cdots))$ to ease presentation. When the context is clear we also omit the subscript $\mathcal{D}_\mathcal{U}$ and simply use σ to denote $\sigma_{\mathcal{D}_\mathcal{U}}$.

Estimate(pw, σ) : This interface returns the estimated frequency of a password pw based on the given count sketch state σ.

To implement DALock with high accuracy, we want the estimator to have the following correctness property: $\mathsf{Estimate}(pw, \sigma) \approx \mathsf{F}(pw, \mathcal{D}_\mathcal{U})$, where $\mathsf{F}(pw, \mathcal{D}_\mathcal{U})$ denotes the actual frequency of pw in $\mathcal{D}_\mathcal{U}$.

TotalFreq(σ) : This operation returns the total number of passwords based on state σ.

Based on the above definition, we denote the *estimated popularity* of a password pw by σ with $\mathsf{p}(pw, \sigma) = \frac{\mathsf{Estimate}(pw,\sigma)}{\mathsf{TotalFreq}(\sigma)}$. For the rest of the discussion, we sometimes omit σ when there is no ambiguity to simplify the presentation. e.g. $\mathsf{p}(pw) = \mathsf{p}(pw, \sigma)$. In addition, we allow the above APIs to take a set of passwords as an argument and return the summed results. i.e. $\mathsf{p}(S) = \sum_{pw \in S} \mathsf{p}(S)$.

$\sigma_{dp} \leftarrow \mathbf{DP}(\epsilon, \sigma)$: This function outputs an ϵ-differentially private count sketch state σ_{dp} constructed from σ with privacy budget ϵ (See **Section** 3.2 for differential privacy).

3.2 Differential Privacy

Differential privacy [14] is a compelling mathematical definition of privacy that has begun to see industrial deployment[15]. It is often viewed as a gold standard for data privacy. In this work, we adopt differentially private count sketches to reduce the risk of privacy leakage. Based on our notion of count sketch, one can define differential privacy as follows.

Definition 1 (ϵ-Differential Privacy [14]). *A randomized mechanism $\mathcal{M}$ gives ϵ-differential privacy if for any pair of neighboring datasets $\mathcal{D}_{\mathcal{U}}$ and $\mathcal{D}'_{\mathcal{U}}$, and any $\sigma \in Range(\mathcal{M})$,*

$$\Pr[\mathcal{M}(\mathcal{D}_{\mathcal{U}}) = \sigma] \le e^{\epsilon} \cdot \Pr\left[\mathcal{M}(\mathcal{D}'_{\mathcal{U}}) = \sigma\right].$$

We consider two datasets $\mathcal{D}_{\mathcal{U}}$ and $\mathcal{D}'_{\mathcal{U}}$ to be neighbors i.f.f. either $\mathcal{D}_{\mathcal{U}} = \mathcal{D}'_{\mathcal{U}} + pw_u$ or $\mathcal{D}'_{\mathcal{U}} = \mathcal{D}_{\mathcal{U}} + pw_u$, where $\mathcal{D}_{\mathcal{U}} + pw_u$ denotes the dataset resulted from adding the tuple pw_u(a new password) to the dataset $\mathcal{D}_{\mathcal{U}}$. We use $\mathcal{D}_{\mathcal{U}} \simeq \mathcal{D}'_{\mathcal{U}}$ to denote two neighboring datasets. This protects the privacy of any single tuple(password) because adding or removing any single password results in e^{ϵ}-multiplicative-bounded changes in the probability distribution of the output. If an adversary can make certain inference about a password based on the output, then the same inference is also likely to occur even if the tuple does not appear in the dataset.

Laplace Mechanism The Laplace mechanism is a classic tool to achieve differential privacy. It computes a differentially private state σ based on dataset $\mathcal{D}_{\mathcal{U}}$ by adding random Laplace noise. The magnitude of the noise depends on GS_{σ}, the *global sensitivity* or the L_1 sensitivity of σ. GS_{σ} quantifies the maximum impact on σ if one adds or removes any record.

Differentially Private Count Sketch Given a CS state σ, adding (removing) any password pw to(from) σ can result in at most d + 1 changes for l_1 norm. Because each pw contributes to d entries in the $d \times w$ table CS.ARRAY and total count CS.T. Therefore, To release σ with privacy budget ϵ, it suffices to add Lap($\frac{d+1}{\epsilon}$) to all entries in σ.

Differential Privacy in Passwords Naor et al.[28] designed a locally differentially private mechanism to identify the most popular passwords in a distribution. Blocki et al. [2] developed a differentially private mechanism for integer partitions and used this to release a private summary of the Yahoo! password dataset. StopGuessing[34] uses a binomial ladder to identify "heavy hitters" (popular passwords), though the data-structure does not provide any formal privacy guarantees such as differential privacy. The data-structure is not suitable for DALock as it provides a binary classification, i.e., either the password is a "heavy hitter" or it is not. For DALock requires a more fine-grained estimate of a password's popularity.

3.3 Notation Summary

In this section, we summarize frequently used notations in this paper across all sections in **Table** 2, Appendix. For a password $pw \in \mathcal{P}$, we use $\mathsf{P}(pw)$ to denote the probability each user selects the password pw. We assume that there is some underlying distribution over user passwords and use $\mathsf{P}(pw)$ to denote the probability of the password $pw \in \mathcal{P}$. It will be convenient to assume that all passwords $\mathcal{P} = \{pw_1, pw_2, \ldots\}$ are sorted in descending order of probability, i.e., so that $\mathsf{P}(pw_1) \ge \mathsf{P}(pw_2) \ldots$.

We use $\mathcal{U} = \{u_1, \ldots, u_N\}$ to denote a set of N users and $\mathcal{D}_{\mathcal{U}} \subseteq \mathcal{P}$ is a multiset of user passwords, i.e., $\mathcal{D}_{\mathcal{U}} = \{pw_{u_1}, \ldots, pw_{u_N}\}$. We typically view $\mathcal{D}_{\mathcal{U}}$ as N independent samples from an underlying distribution over $\mathcal{P}$ and write $\mathsf{F}(pw, \mathcal{D}_{\mathcal{U}}) = \left|\{i : pw_{u_i} = pw\}\right|$ to denote the number of times the password pw was observed in our sample. We often omit $\mathcal{D}_{\mathcal{U}}$ in the notation when the dataset is clear from the context and simply write $\mathsf{F}(pw)$.

We remark that $\mathsf{P}(pw) = \frac{\mathbb{E}[\mathsf{F}(pw, \mathcal{D}_{\mathcal{U}})]}{N}$ and thus for popular passwords we expect that the estimate $\mathsf{P}(pw) \approx \frac{\mathsf{F}(pw, \mathcal{D}_{\mathcal{U}})}{N}$ will be accurate for sufficiently large N. However, because the underlying password distribution is unknown and an authentication server cannot store a plaintext encoding of $\mathcal{D}_{\mathcal{U}}$ we will often use other techniques to estimate $\mathsf{P}(pw)$ and/or $\mathsf{F}(pw, \mathcal{D}_{\mathcal{U}})$. In particular, we consider a count sketch data structure CS trained on $\mathcal{D}_{\mathcal{U}}$ (or a small subsample of $\mathcal{D}_{\mathcal{U}}$), which allows us to generate an estimate $\mathsf{p}(pw)$ for the popularity of each password. Similarly, we can also use password strength meters to compute $\mathsf{p}(pw)$ to estimate $\mathsf{P}(pw)$.

4 THE DALock MECHANISM

In this section, we present the DALock mechanism, discuss how DALock might be implemented, and the strategies an attacker might use when DALock is deployed. Intuitively, DALock punishes incorrect password guesses more harshly since an attacker will want to submit popular password guesses to maximize their chances of cracking the users' passwords.

4.1 DALock

The K-strikes mechanism keeps track of a single parameter K_u for each user u, which represents the number of consecutive incorrect login attempts on u's account. Each failed login attempt makes K_u increment by 1. In contrast, successful authentication resets K_u to 0. If we ever have $K_u \ge K$, then the throttling mechanism kicks in, and the authentication server will lock down the account until the user takes corrective action[1].

The key-idea behind DALock is to additionally maintain an extra "hit count" variable Ψ_u for each user u. Intuitively, Ψ_u measures the total probability mass of all incorrect guesses submitted on u's account. Initially, when a new user u registers, we will have Ψ_u= 0 (and K_u= 0). After each failed attempt with an incorrect password $pw \ne pw_u$, the hit count variable Ψ_u and strike count variable K_u will be increased by $\mathsf{p}(pw)$ and 1, respectively. i.e., Ψ_u += $\mathsf{p}(pw)$, and K_u += 1. Here, $\mathsf{p}(pw)$ denotes an estimate of the probability of the password pw. Incorrect passwords are punished more severely when pw is an overly popular password. Upon successful authentications, Ψ_u never resets, unlike the consecutive strike parameter K_u. DALock throttles u's account if the "hit count" exceeds Ψ (i.e., $\Psi_u \ge \Psi$) or if there are too many consecutive mistakes (i.e., $K_u \ge K$). For example, suppose that the (estimated) probability of the passwords "aaa," "bbb," and "ccc" were 3%, 1.7% and 0.8%,

[1] For example, the user might be asked to resetting their password via e-mail or wait for some fixed amount of time. In some settings, the user might simply be asked to solve a CAPTCHA challenge. The latter approach has some usability advantages and security drawbacks, e.g., a malicious attacker might pay human to solve the CAPTCHA challenges so that they can continue attempting to guess the user's password.

respectively. If a user registers with password "ddd" and then attempts to login with the previous three passwords, Ψ_u will be set to $0.055 = 0.03 + 0.017 + 0.008$.

Each time the user (or attacker) attempts to login with a password pw the response from the authentication server will either be (1) "locked" if $\Psi_u \geq \Psi$ or if $K_u \geq K$, (2) "correct" if the guessed password matches the user password i.e., $pw = pw_u$[2], or (3) "incorrect password" otherwise. We remark that the authentication server could intentionally blur this distinction between cases (1) and (3), but this comes at a usability cost, e.g., an honest user would be annoyed if they were repeatedly informed that their password is incorrect when the account is actually locked.

Remark: One could optionally consider initializing the hit count parameter Ψ_u based on the strength of the user's password. For example, if u registers with a weak password, then we might initialize $\Psi_u = \Psi/2$ for stronger protection, i.e., so that the account is locked down faster. Similarly, a user with a strong password might be awarded by setting $\Psi_u = \Psi$. However, because Ψ_u and K_u are stored on the authentication server, this would leak information about the strength of pw_u to an offline attacker, e.g., if an offline attacker sees that $\Psi_u = \Psi/2$, they might reasonably infer that the user picked a weak password. [3]

4.2 DALock **Authentication Server**

To implement DALock, we need an efficient way to estimate the probability $\mathrm{p}(pw)$ of each incorrect password pw. We consider several instantiations of this frequency oracle. One option is to use password strength meters such as ZXCVBN [46] or more sophisticated password cracking models [27, 38]. e.g., Markov Models, Probabilistic Context-Free Grammars, or Neural Network. Another naive approach would be to maintain a plaintext list of all user passwords along with their frequencies. However, this approach is inadvisable due to the risk of leaking this plaintext list. Herley and Schechter [33] proposed using the count sketch data-structure, which would allow us to estimate the frequency of each password without explicitly storing a plaintext list. However, there are no formal privacy guarantees for this approach. We chose to adopt a differentially private count sketch to address privacy concerns. The authentication server initializes the count sketch $\sigma_{dp} \leftarrow \mathrm{DP}(\epsilon, \sigma)$ by adding Laplace Noise to preserve ϵ-differential privacy. Each time a new user u registers with a new password pw_u, it would be added to the count sketch.

We remark that maintaining a differentially private count sketch has many other potentially beneficial applications, e.g., one could use the count sketch to ban weak passwords [33] and/or to help identify IP addresses associated with malicious online attacks [34]. One disadvantage is that the attacker will also be able to view the count sketch if the data structure is leaked. The usage of differential privacy helps to minimize these risks. Intuitively, differential privacy hides the influence of any individual password, ensuring that an attacker will not be able to use the count sketch data-structure to help identify any unique password. However, an attacker may still be able to use the data-structure to learn that a particular password is globally popular (without linking that password to a particular user). We argue that this is not a significant risk as most attackers will already know about globally popular passwords, e.g., from prior breaches.

5 EXPERIMENTAL DESIGN

We evaluate the performance of DALock through an extensive battery of empirical simulations. In this section, we describe the modeling choices we made when designing our experiments. To simulate the authentication ecosystem, we need to simulate honest users' behavior, the authentication server running DALock, and an online attacker.

Briefly, when simulating users, we need to model the distribution over users' passwords, the distribution over honest login mistakes (e.g., typos or recall errors), and the user's login schedule. When simulating the distribution over users' passwords, we use multiple empirical datasets to define the underlying password distribution. We use a Poisson arrival process to model the frequency of user login attempts [1]. Our model for users' mistakes is informed by recent empirical studies of password typos [11, 12] and is augmented to simulate other mistakes, i.e., recall errors. The key question for simulating an authentication server running DALock is how the (password) frequency oracle $\mathrm{p}(\cdot)$ is implemented. We consider two concrete implementations: password strength models [27, 38, 46] (e.g., ZXCVBN, Markov Models, Neural Networks) and (differentially private) count sketches. When simulating the attacker, we consider an untargeted one who knows the distribution over user passwords as well as the DALock mechanism — including the frequency oracle $\mathrm{p}(\cdot)$. We leave the question of tuning DALock to protect against targeted online attackers [44] as an important direction for future research. We elaborate on each of these key model components below. We begin by with an overview of the empirical datasets $\mathcal{D}_{\mathcal{U}}$ that we used in our experiments.

5.1 Experimental Datasets

In this work, we use multiple real-world password datasets (see Table 1). Those datasets were either hacked or leaked via various vulnerabilities and eventually made public on the Internet. The only exception is the Yahoo dataset, which is a sanitized password frequency dataset collected [5] with permission from Yahoo!. It consists of anonymized password histograms representing almost 70 million Yahoo! users who logged into their account during a 48-hour window in May 2011. Yahoo! later authorized the public release of a differentially private version of this dataset [2]. We remark that this frequency corpus *does not contain any plaintext passwords*, so we did not use password strength models in our experiments involving the Yahoo! dataset.

Each dataset defines an empirical password distribution. In each of our experiments, we assume that this distribution matches the real (unknown) password distribution from which these datasets were sampled. While the empirical distribution may not precisely match the real one, we stress that our analysis focuses on the most popular passwords in the distribution — the ones that an attacker will try to guess. Because the datasets are all quite large (the smallest dataset has over 0.5 million passwords), standard concentration bounds imply that the true probability of a popular

[2] To ease presentation, we omit the description of the password hashing algorithm when we describe the authentication server. In practice, we recommend that the authentication server only stores salted password hashes using a moderately expensive key derivation function to increase guessing costs for an offline attacker.

[3] One could potentially avoid storing Ψ_u unencrypted if one is willing to implement a silent lockout policy where the user cannot distinguish between an incorrect guess and a locked account, but we wish to avoid solutions that blur this distinction.

password in the distribution will almost certainly closely match the empirical probability.

Dataset	Passwords	Accounts	$\mathrm{P}(pw_1)$	$\mathrm{P}(pw_{1-10})$
Yahoo	33,895,873	69,301,337	1.1%	1.9%
RockYou	14,341,564	32,603,388	0.89%	2.1%
000webhost	10,587,915	14,960,642	0.081%	0.48%
LinkedIn	6,840,885	68,361,064	1.53%	2.82%
CSDN	4,037,268	5,908,494	1.29%	3.72%
clixsense	1,628,297	2,195,900	0.15%	0.7%
brazzers	587,934	925,614	0.58%	1.13%
bfield	416,034	539,434	0.48%	1.97%

Table 1: Summary of dataset

Ethics: The datasets we used contain passwords that were previously stolen and subsequently leaked online. The use of such data raises critical ethical considerations; however, such password lists are already publicly available online, so our use of the data does not exacerbate the prior harm to users. We did not crack any new user passwords. Furthermore, the datasets we use have been cleaned of all identifying information beyond the passwords themselves. In summary, we believe that our use of the leaked data will not exacerbate prior harm to users, and the lockout mechanism we develop and evaluate may help to protect user passwords in the future.

5.2 Modeling Users

Our model to simulate honest users' behavior consists of three key components: user password selection, login frequency, and mistake model.

5.2.1 Simulating Users' Password Choices. In each simulation, we fix a dataset that is used to simulate user password selection. In particular, a dataset consists of a multiset $\mathcal{D}_{\mathcal{U}} = \{pw_1, \cdots, pw_N\}$ of N passwords which can be compressed into pairs $(pw, \mathsf{F}(pw, \mathcal{D}_{\mathcal{U}}))$ where $\mathsf{F}(pw, \mathcal{D}_{\mathcal{U}})$ denotes the number of times the password pw occurs in the dataset $\mathcal{D}_{\mathcal{U}}$. Each dataset $\mathcal{D}_{\mathcal{U}}$ induces an empirical distribution over users' passwords where the probability of sampling each password pw is simply $\frac{\mathsf{F}(pw,\mathcal{D}_{\mathcal{U}})}{N}$. Each simulated user u in our experiment has 6 different passwords sampled from this empirical distribution and registers with the first sampled password. The remaining five extra sampled passwords will be used to help to simulate recall errors (see **Section** 5.2.3), e.g., they represent the user's passwords for other websites.

Ban-list We additionally consider the setting where the authentication server chooses to ban users from selecting the top B passwords, e.g., top 10 passwords. We use the normalized probabilities model [4] to simulate users' password selections under this restriction. In this model, we simply use rejection sampling to avoid sampling one of the top B passwords. Equivalently, we can let $\mathcal{D}_{\mathcal{U},B}$ denote the dataset $\mathcal{D}_{\mathcal{U}}$ with the B most common passwords removed and sample from the empirical distribution corresponding to the updated dataset $\mathcal{D}_{\mathcal{U},B}$.

5.2.2 Simulating User's Login Patterns. To simulate users, we need to model the frequency with which our honest user attempts to login to the authentication server. In particular, we aim to simulate the login behaviors over a 180-day time span. For each user u, we want to generate a time sequence $0 < t_1^u < t_2^u < \cdots < 4320 = 180 \times 24$ where each $t_i^u \in \mathbb{N}$ represents the time (in hours) of the ith user visit. Following prior works (e.g., see [1, 23]), we use a Poisson arrival process to generate the sequence. The Poisson arrival process is parameterized by an arrival rate T_u (hours), which encodes the expected time between consecutive login attempts $T_u = \mathbb{E}[t_{i+1} - t_i]$. The arrival process is memoryless, so the actual gap $t_{i+1} - t_i$ is independent of t_i. Since some users are more active than others, we pick a different arrival rate T_u for each user u where each T_u is sampled uniformly at random from $\{12, 24, 24 \times 3, 24 \times 7, 24 \times 14, 24 \times 30\}$. The parameter $T_u = 12$ (hours) corresponds to users who login to their accounts twice per day on average, while the parameter $T_u = 24 \times 30$ corresponds to a user who visits the site once per month. We assume that users continue attempting to login for each user visit until they succeed or get locked out.

To independently study the throttling effects of DALock, we do not simulate users who completely forget their passwords as these users will need to reset their passwords independently of the deployed throttling mechanism. In addition, we do not simulate a client device that automatically attempts to login on the user's behalf. It may be desirable to have the authentication server stores the (salted) hash of the user's previous password(s) to avoid locking the user's account in settings where a client device might repeatedly attempt to login with an outdated password. Alternatively, the authentication server could store an encrypted cache of failed login attempts using public-key cryptography. Each failed login attempt $pw'_u \neq pw_u$ would be encrypted with a public key pk_u and stored on the authentication server. The encrypted cache could only be decrypted when the user authenticates with the correct password[4]. The encrypted cache could be used as part of a personalized typo corrector [12] and could also be used to avoid penalizing repeat mistakes [12, 34]. One potential downside to this approach is that the cache might inadvertently contain credentials from other user accounts, making cached data valuable to the attacker. More empirical studies would be needed to determine the risks and benefits of maintaining such a cache.

5.2.3 Simulating Users' Mistakes. The last component of our user model is a mechanism to simulate users' honest mistakes during the authentication process. Our model relies upon recent empirical studies of password typos [11, 12] and additionally incorporates other common user mistakes, e.g., recall errors. The aforementioned studies show that roughly 7.5% of login attempts are mistakes, and at least 68% of them are (most likely) typos, i.e., within editing distance 2 of the original passwords.

Accordingly, we set the mistake rate to be 7.5% for simulation. When simulating each login attempt, the user will enter the correct password with probability 92.5%. Otherwise, the user will enter an incorrect password with probability 7.5% and the next step is to simulate the error(s). Based on the statistics mentioned earlier, we simulate typos and recall errors with probability 68% and 32%, respectively. To simulate a recall error, we randomly select one of the user's five alternate passwords to model a user who forgot which

[4] Unlike the public encryption key pk_u, which would be stored on the authentication server, the secret key sk_u would only be stored in encrypted form i.e., the server would store $c_u = \mathbf{Enc}_{K_u}(sk_u)$ where $K_u = \mathbf{KDF}(pw_u)$ is a symmetric encryption key derived from the user's password.

of their passwords was associated with this particular account. If the user recalls the wrong password, they might additionally miss-type it (with probability $0.075 \cdot 0.68$). We refer an interested reader to **Appendix** C for a more detailed discussion of our mistake model, including a flow chart (see Figure 6) and more fine-grained typo statistics.

5.3 Modeling the Authentication Server

We model an authentication server running (K, Ψ)-DALock with various K and Ψ settings. Each time a user u (or attacker pretending to be u) failed to login, the authentication server updates the parameters Ψ_u and K_u accordingly following the DALock mechanism. Notice that when $\Psi = \infty$, the authentication server is actually running the classical K-strikes lockout policy. To deploy DALock with a finite hit-count parameter Ψ, an authentication server needs to use a frequency oracle to update the hit count after each failed login attempt. In this work, we consider two concrete approaches the authentication server might adopt: (differentially private) count sketch estimator and password strength models. We use $\mathsf{p}(pw, \mathsf{Estimator})$ to denote the estimated popularity (probability) of a password pw estimated by the estimator Estimator, e.g., given a count sketch σ we would use $\mathsf{p}(pw, \sigma) = \frac{\mathbf{Estimate}(pw,\sigma)}{\mathbf{TotalFreq}(\sigma)}$.

5.3.1 Differentially Private Count Sketch Estimator. The first instantiation of $\mathsf{p}(\cdot, \cdot)$ we consider is to build a count sketch estimator $\sigma_{\mathcal{D}_\mathcal{U}} = \mathsf{Add}(\mathcal{D}_\mathcal{U}, \sigma)$ from the dataset $\mathcal{D}_\mathcal{U}$ directly. The authentication server would update the count sketch with the new password each time a new user registers. [5]. When deploying the count sketch estimator, there are several issues to consider: memory efficiency, privacy, sample size, and accuracy.

Memory Efficiency We instantiate the count sketch with parameters $d = 5$ and $w = 10^6$ so that the entire data structure requires just 20 MB of space, which easily fits in modern RAM.

Privacy As we discussed earlier, one concern about storing a count sketch $\sigma_{\mathcal{D}_\mathcal{U}}$ on the authentication server is that an offline attacker might steal this file and use the data-structure to help identify users' passwords. For example, if our user John Smith selects (resp. does not select) a rare password "J.S.UsesStr0ngpwd!" then we would expect that the true frequency of this password is $\mathsf{F}(pw, \mathcal{D}_\mathcal{U}) = 1$ (resp. $\mathsf{F}(pw, \mathcal{D}_\mathcal{U}) = 0$). If the count sketch estimator is overly accurate, then the attacker would be able to learn that one user (most likely John Smith) picked this password. Without a way to address these privacy concerns, an organization might be understandably wary of deploying a count sketch estimator.

To address these privacy concerns, we consider an ϵ-differentially private estimator $\sigma_{dp} = \mathbf{DP}(\epsilon, \sigma)$ in our experiments. During initialization, we add Laplace noise to the count sketch where the noise parameter scales with $\frac{d}{\epsilon}$. In our above example, differential privacy ensures that — up to a multiplicative advantage e^ϵ — an attacker cannot use the count sketch to distinguish between a dataset in which John Smith did (resp. did not) pick the password "J.S.UsesStr0ngpwd!'. Notice that lower values of ϵ correspond to stronger privacy guarantees and we can use $\epsilon = \infty$ to indicate no differential privacy guarantee. In most of our experiments, we use small privacy parameters $\epsilon = 0.1$, which is much smaller than the privacy parameters used in most prior deployments of differential privacy, e.g., $\epsilon = 0.5$ for releasing Yahoo! password corpus[2], $\epsilon \geq 2$ for collecting users' information [36], and $\epsilon \geq \ln 81$ for RAPPOR [15, 45].

Sample Size and Accuracy In general, the accuracy of a count sketch increases with the size of the password dataset. Suppose that the organization does not have millions of users or the dataset size is decreased because it allows users to "opt-out" of the data collection. One natural question is whether one would be able to deploy a count sketch to obtain reliable frequency estimates under such circumstances. We investigate this question by subsampling smaller datasets to train the count sketch. Given a set $\mathcal{U}$ of N users, we use $\mathcal{U}_{r\%}$ to denote a randomly subsampled set of $r\%$ of users. We use $\mathcal{D}_{\mathcal{U}_{r\%}}$ to denote the corresponding subsampled password dataset and $\sigma_{r\%} = \mathsf{Add}(\mathcal{D}_\mathcal{U}, \sigma)$ to denote the count sketch trained on the subsampled data. The question is whether $\sigma_{r\%}$ can be as effective as σ for deploying DALock.

In our experiments, we consider the following sampling rates: 1%, 5%, and 10%. Our empirical results show that using approx. 0.3 million passwords is sufficient to train a reliable count sketch. A substantially small sample like 1% rate can hurt the performance of count sketch, especially when the original dataset $\mathcal{D}_\mathcal{U}$ is already small. (e.g., bfield). On the positive side, if one picks an adequate sampling rate r (e.g., 10%) or the original dataset size is sufficiently large (e.g., 000webhost), then $\sigma_{r\%}$ can perform nearly as good as σ.

Count Sketch with Ban-list In our simulations, we also consider an authentication server that bans a list of popular passwords from the dataset to help flatten the password distribution and protect users against online attacks. Theoretical analysis indicates that directly banning the most popular passwords is one of the most effective ways to increase the minimum entropy of the password distribution [4]; On the other hand, banning too many of them may raise a usability concern – a large portion of users need to pick their new passwords (see **Figure** 2). One additional benefit of using a count sketch data structure is that it can be used to help implement such policy, i.e., if a user attempts to register with password pw and $\mathsf{p}(pw, \sigma)$ is already too high, then the user will be asked to pick a different password [33].

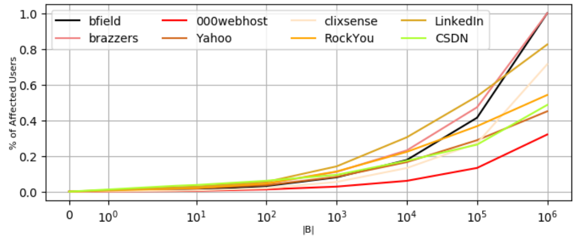

Figure 2: Affected Users vs Ban-List size

We evaluate the performance of DALock in the presence of various sizes of ban lists. Recall that we let $\mathcal{D}_{\mathcal{U},B}$ denote the dataset $\mathcal{D}_\mathcal{U}$ with the B most common passwords removed. To model how affected users will update their passwords in response to the ban-list, we follow the normalized probabilities model of [4]. In particular, we assume users who are affected by the policy will pick new passwords following the empirical distribution induced by $\mathcal{D}_{\mathcal{U},B}$. We then train the count sketch based on the updated dataset, i.e., $\sigma_{\mathcal{D}_{\mathcal{U},B}} = Add(\mathcal{D}_{\mathcal{U},B})$.

[5] The count sketch instantiations we consider would also support a remove operation which would allow the authentication server to handle password updates efficiently

5.3.2 Frequency Oracle from Password Models. As we previously discussed, there are several reasons why an organization might prefer not to use a count sketch for frequency estimation, e.g., privacy concerns or insufficient users. One alternative approach is to instantiate the frequency oracle with a password model. This could be a heuristic password strength meter, a more sophisticated model based on Neural Networks, Probabilistic Context-Free Grammars, Markov Models, or an empirical estimate based on Hashcat. The primary advantage of this approach is that the model can be deployed immediately even before an organization has any users and there are no privacy concerns.

We adopted the ZXCVBN password strength meter [46] as prior empirical studies demonstrate that it is one of the most accurate password strength meters [20]. In addition, we used the Password Guessing Service (PGS) [27, 38] to obtain guessing numbers for Neural Network, PCFG, Hashcat, and Markov Models — we also considered the minimum guessing number across all four models as suggested in [38]. For example, if a password pw had a guessing number g, we might estimate that $\mathrm{p}(pw_i) = 1/g$. One challenge we need to address is that the estimates we obtain do not always yield a probability distribution. E.g., for ZXCVBN we have $\sum_{i=1}^{10000} \mathrm{p}(pw_i) \gg 1$ where i ranges over the top 10^4 remaining passwords in the dataset. Thus, before deploying the frequency estimator for DALock, we renormalized our estimates so that $\sum_{i=1}^{\max\{10^4, B\}} \mathrm{p}(pw_i) = 1$ where B is the number of banned passwords. [6]

5.4 Modeling the Attacker

The final component of our simulation is a model of the attacker. We take a conservative approach and model an untargeted attacker with complete knowledge of the password distribution. Following Kerckhoff's principle, we also assume that the attacker has access to the complete description of the DALock mechanism (e.g., K and Ψ). In particular, for any password pw, we assume that the attacker knows both the true probability $\mathrm{P}\,(pw)$ and the estimated probability $\mathrm{p}(pw)$. We also assume that the attacker is given the complete sequence of login times $t_1^u \leq t_2^u \leq \ldots \leq 24 \times 180$ for each user u over a 180-day time span as well as the outcome of each, e.g., at time t_i^u user u will login successfully after 2 incorrect attempts. Finally, we assume the attacker can infer the strike threshold and hit count threshold for any user u at any time t because they are given the complete sequence of login times and outcomes. We use $K_{u,t}$ (resp. $\Psi_{u,t}$) to denote the strike (resp. hit count) threshold on user u's account at time t, assuming that the attacker does not submit any of their own guesses.

Remark: We conservatively aim to overestimate the capabilities of an untargeted online attacker. In practice, the online attacker will be able to approximate $\mathrm{P}\,(pw)$ and $\mathrm{p}(pw)$ over time by interacting with the DALock server, e.g., by setting up dummy accounts to test many times. Similarly, the attacker would not necessarily know/predict the exact login times and outcomes for a user. However, this conservative assumption makes it feasible to precisely characterize the optimal behavior of an attacker. In practice, an online attacker might wait several days in between guesses to avoid accidentally locking the user's account based on the number of consecutive incorrect login attempts.

5.4.1 Optimizing Attack Strategies. The attacker aims to maximize the probability of cracking each password within the fixed 180-day time span. For example, the attacker might try to find a popular password pw where the ratio $\frac{\mathrm{p}(pw)}{\mathrm{P}(pw)}$ is small so that the increased hit count is smaller than intended when it fails. We formalize the attacker's optimal strategy in terms of the Password Knapsack problem (PK). Unsurprisingly, the password knapsack problem turns out to be NP-hard(see **Appendix** A), but there are several heuristic algorithms the $\mathcal{A}$ can use to achieve nearly optimal results in practice.

Supposing that the attacker wishes to avoid locking down the user's account before a particular time t, then the cumulative (estimated) probability of all guesses submitted before that time should be at most $\Psi'_{u,t} := \Psi - \Psi_{u,t}$. Similarly, we let $M(t)$ denote the maximum number of guesses that the attacker can sneak in over the first t hours without locking down the account, i.e., because $K_{u,t'} \geq K$ at some time $t' \leq t$. (Recall K_u resets whenever u login successfully).

Fixing a time parameter t, the attacker's goal is to find a subset $S_t \subseteq \mathcal{P}$ of $M(t)$ passwords to guess such that

$$\sum_{pw \in S_t} \mathrm{p}(pw) \leq \Psi'_{u,t} \ . \tag{1}$$

After checking the passwords in S_t the attacker can still guess one more password $pw_{hold} \notin S_t$ before the account is locked down. Given a set S_t and a holdout password $pw_{hold} \notin S_t$ the probability that the attacker succeeds is

$$\mathrm{P}\,(pw_{hold}) + \sum_{pw \in S_t} \mathrm{P}\,(pw) \ . \tag{2}$$

Thus, the goal of the attacker is to find a subset S_t of size $|S_t| \leq M(t)$ maximizing their success rate (equation 2) subject to the constraints in equation 1.

Password Knapsack Problem Given a password dictionary $\{pw_1, \ldots, pw_n\}$ we formally define the Password Knapsack(PK) problem as the following integer program with indicator variables $s_i \in \{0, 1\}$ and $l_i = \{0, 1\}$ for each password pw_i. The attackers goal is to select a holdout password and a separate subset of M $(= M(t))$ passwords with total 'weight' (hit count) at most Ψ' $(= \Psi'_{u,t})$

$$\max \sum_i (s_i + l_i) \cdot \mathrm{P}\,(pw_i)$$

$$\textit{subject to,} \quad \begin{array}{rr} \sum_i s_i \cdot \mathrm{p}(pw_i, \sigma)) \leq \Psi' & \sum_i s_i \leq M \\ \sum_i l_i \leq 1 & \forall i \; l_i + s_i \leq 1 \end{array}$$

$$\textit{where,} \quad \forall i, s_i, l_i \in \{0, 1\}$$

Intuitively, setting $s_i = 1$ means pw_i is selected to be placed in the "password knapsack" $S \subseteq \mathcal{P}$, i.e., to be used for dictionary attack. Setting $l_i = 1$ indicates that password pw_i is used as a holdout password. The constraints ensure that $|S| \leq M$ and we pick exactly one holdout password that is not already in S.

Solving the Password Knapsack To maximize the number of cracked passwords, an online attacker can compute $M(t)$ and $\Psi'_{u,t} :=$

[6] We estimate $\sum_{i=1}^{\max\{B\}} \mathrm{p}(pw_i)$ by sampling 20,000 users' passwords from $\mathcal{D}_{\mathcal{U},B}$ when B $\geq 10^5$ to avoid submitting too many requests to PGS.

$\Psi - \Psi_{u,t}$ for each time $t \leq 24 \times 180$ and solve the corresponding Password Knapsack problem. Given optimal solutions $(pw^*_{hold,t}, S^*_t)$ for each time t, the attacker will pick the solution that maximizes the number of cracked passwords as in equation 2. Notice that the calculations above need to be *repeated for each user u* since the values $M(t)$ and $\Psi'_{u,t}$ may vary due to different visitation schedules.

The Password Knapsack problem is NP-hard as we prove in the Appendix(**Theorem** A.1) via a straightforward reduction from Subset Sum. In all of the instances, we considered we found that the holdout password's optimal choice was simply pw_1, the most likely password in the distribution. Once we fix our holdout password, our problem reduces to the two-dimensional knapsack problem. Assuming $P \neq NP$ the two-dimensional knapsack problem does not even admit a polynomial-time approximation scheme (PTAS) [25] in contrast to the regular knapsack problem, which has a fully polynomial-time approximation scheme (FPTAS)). Thus, we consider two heuristic approaches to solve PK: Dantizig's Algorithm Based[13] approach (DAB) and Feasible Most Promising Password First approach(FMPPF).

DAB sorts passwords $\mathcal{P}_{\tilde{\Pi}} = \{pw_2, \dots pw_n\}$ based on the how much they are *underestimated*, i.e., $\frac{\mathsf{P}(pw_i)}{\mathsf{p}(pw_i)}$, and selects guesses based on such sorted order until either M passwords are selected or adding the next password to the knapsack would exceed capacity Ψ'. FMPPF sorts the passwords differently by using the true probability $\mathsf{P}(pw_i)$ and FMPPF simply selects password pw in sorted order. More detailed discussion can be found in **Appendix** B. Intuitively, FMPPF (resp. DAB) will perform better when M (resp. Ψ') is the (major) limiting constraint.

We found that FMPPF generally performs better than DAB despite its simplicity. Besides, our simulation shows that FMPPF's performance is close to optimal. Practically speaking, one generally expects $\mathsf{p}(pw_i) \approx \mathsf{P}(pw_i)$, especially when pw_i is a popular password. Thus, DAB can hardly gain advantages from underestimation. Furthermore, imagine one bucket of passwords by probability ranges, there are plenty of passwords in each bucket. Intuitively, picking passwords ordered by $\mathsf{P}(pw_i)$ should produce an (almost) optimal solution (quickly). Thus, we choose to present the results based on the FMPPF approach.

6 EXPERIMENTAL RESULTS

We empirically evaluated the performance of DALock under a variety of scenarios. During each simulation, we had 10^6 honest users registered on an authentication server running DALock. We simulate their login behaviors (see section 5.2) over a period of 180 days. To analyze usability, we ran simulations without an online password attacker and measured unwanted lockout rate, i.e., the fraction of user accounts locked due to honest mistakes. To analyze security, we added an untargeted online attacker $\mathcal{A}$ (see section 5.4) to the simulation and measured the fraction of user passwords $\mathcal{A}$ cracked. In our simulations, we do not consider other defenses the authentication server might adopt (e.g., banning malicious IPs) since our goal is to focus on the impact of the DALock mechanism.

Figure 3 directly compares the usability/security of DALock for a fixed banlist size $B = 10^4$ as the hit count threshold Ψ varies. Similarly, **Figure** 4 (resp. **Figure** 5) highlights the security (resp. usability) of DALock as the banlist size varies holding the DALock parameters $k = 10$ and Ψ constant. We repeat the simulation instantiating the DALock frequency oracle with a differentially private count sketch, ZXCVBN, HashCat, Markov, Neural Networks, PCFG, and Min (a combination of HashCat, Markov, Neural Networks, and PCFG).

Baseline We used the classical 3-strikes mechanism and the 10-strikes mechanism (recommend by Brostoff et al. [7] to improve usability) as baselines for comparisons. These two mechanisms are equivalent to $(3, \Psi = \infty)$-DALock and $(10, \Psi = \infty)$-DALock respectively.

6.1 Usability/Security Tradeoff

While decreasing the hit count parameter Ψ improves security it also can have an adverse impact on usability. **Figure** 3 directly compares the usability/security of DALock fixing the banlist size $B = 10^4$, $k = 10$ and varying Ψ to measure the % of cracked passwords (resp. % locked users) when the simulation includes (resp. excludes) an online attacker. Legend entries are in the format FrequencyOracle(k) where we fixed the strike parameter $k = 10$ in each of our simulations (excluding the 3-strike mechanism). In the appendix we repeated each simulations with different ban-list sizes to show how DALock performs when the authentication server requires users to pick stronger passwords — e.g., see Appendix, **Figures** 7.

Our results indicate that one can improve *both* security and usability by replacing the classic 3-strikes throttling mechanism with $(10, \Psi) -$ DALock with a properly configured Ψ. **Figure** 3 demonstrates that DALock offers a superior usability/security tradeoff when instantiated with a suitable frequence oracle i.e., 0.1-CS-all and ZXCVBN. Similarly, our results demonstrate that $(10, \Psi)$-DALock achieves comparable usability to classic 10-strikes throttling mechanism while providing much stronger security guarantees.

DALock performs best when instantiated with the differentially private count sketch (0.1-CS-all). We use the notation ϵ-CS-all(resp. ϵ-CS-X%) to refer to an ϵ-differentially private count sketch trained on the entire dataset $\mathcal{D}_{\mathcal{U}}$ (resp. a dataset $\mathcal{D}_{\mathcal{U}_{X\%}}$ obtained by sampling X% of user passwords from $\mathcal{D}_{\mathcal{U}}$). Training the differerentially private count-sketch on 1% of the data is effective for larger datasets such as RockYou and 000webhost, but the usability/security curve is inferior for smaller datasets such as bfield and brazzers. The performance of DALock when instantiated with other frequency oracles is incomparable to the classic 3-strikes mechanism i.e., we can always set Ψ to improve security, but this occasionally results in *inferior* usability.

6.2 Impact of Banlist Size on Security/Usability

We demonstrate the usability/security impact of the ban-list size $B \in \{0, 5, 10, 100, 1000, 10000, 100000\}$ holding the other DALock parameters $k = 10$ and Ψ constant. We restricted our attention to ban-list size $B \leq 10^5$ as larger ones often require more than half of users to change their password in response, e.g., see **Figure** 2 shows that banning 10^5 passwords will already annoy approx. 10% to 50% of users during account creation.

Our main simulation results are summarized in **Figure** 4 (for security) and **Figure** 5 (for usability). The X-axis of each figure corresponds to the ban-list sizes (where $B = 0$ means there is no ban-list). And the Y-axis corresponds to the metric score (compromised

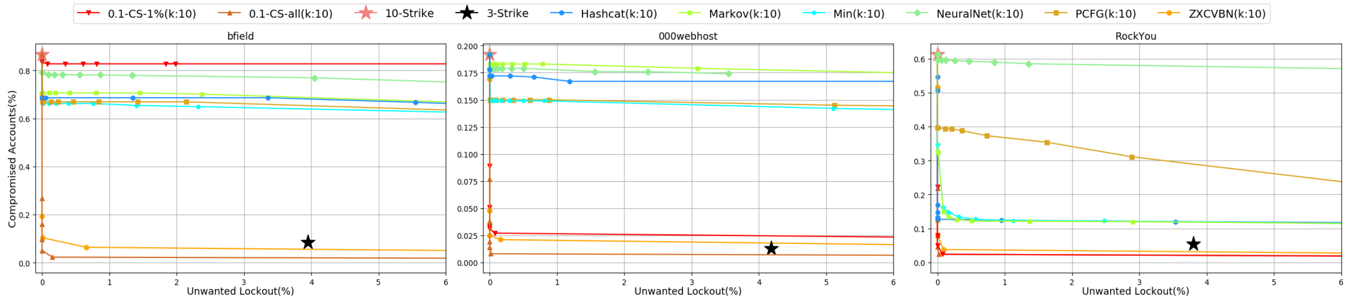

Figure 3: Usability/Security Tradeoff of DALock with $(B = 10^4)$

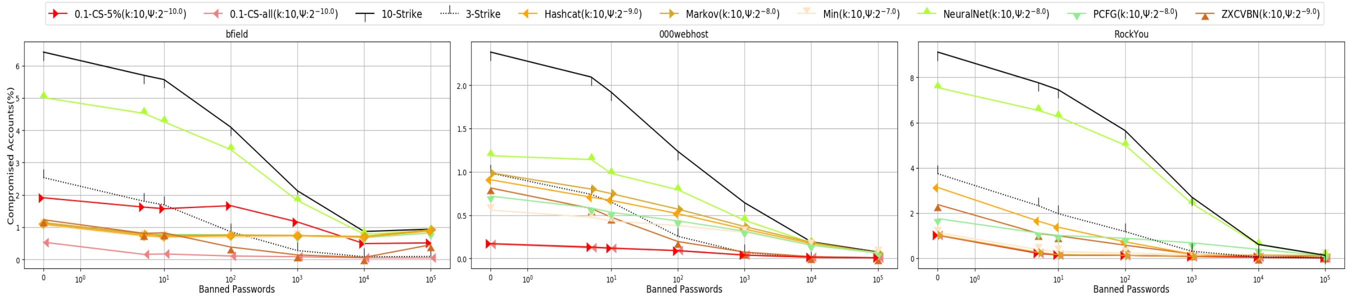

Figure 4: Security Measurement of DALock

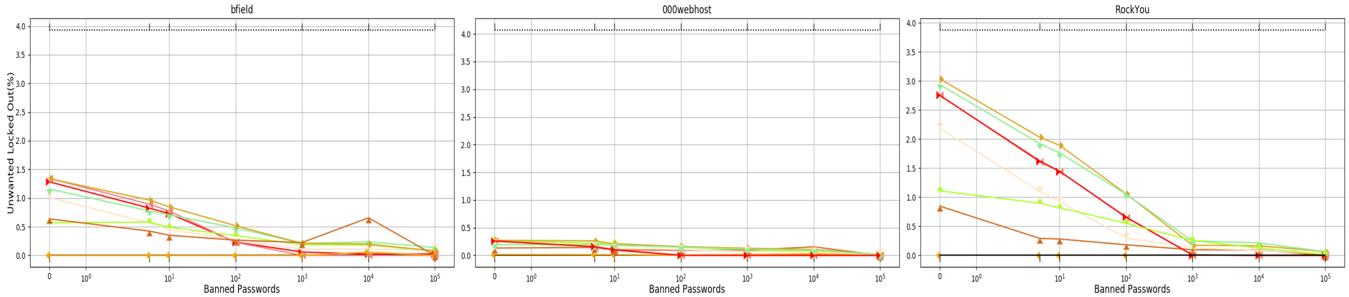

Figure 5: Usability Measurement of DALock

user accounts (%) / unwanted lockout rate (%)) measured after 180 days.

Implementation Details In Figures 4 and 5 we focus on the following (hand-picked) instantiations of DALock: 3-strikes(k:3, Ψ: ∞), 10-strikes(k:10, Ψ: ∞), 0.1-CS-all(k:10, Ψ:$2^{-10.0}$), 0.1-CS-5%(k:10, Ψ:$2^{-10.0}$), ZXCVBN(k:10, Ψ:$2^{-9.0}$), Min(k:10, Ψ:$2^{-7.0}$), Hashcat(k:10, Ψ:$2^{-9.0}$), Markov(k:10, Ψ:$2^{-8.0}$), NeuralNet(k:10, Ψ:$2^{-8.0}$), and PCFG(k:10, Ψ:$2^{-8.0}$). Legend entries are in the format FrequencyOracle(k,Ψ) where k and Ψ are the DALock throttling parameters (with the exception of the 3-strikes mechanism we fixed $k = 10$ in all other simulations). **Figures** 4 and 5 highlight the performance of DALock for handpicked Ψ parameters (e.g., $\Psi = 2^{-10}$ for differentially private count sketch). Additional plots in the appendix explore the impact of the privacy budget ϵ on the Count-Sketch frequency oracle as well the effect of smaller/larger subsampling rates. To save space **Figures** 4 and 5 only show results for the RockYou, 000webhost and bfield datasets while results for the brazzers, csdn and clixsense datasets can be found in the appendix (see **Figures** 8 and 9).

Usability Firstly, **Figure** 5 clearly demonstrates the unwanted lockout rate of (10,Ψ)-DALock is substantially lower than the traditional 3-strikes mechanism. This result held robustly across all datasets irrespective of ban-list size and selection of frequency oracles. For example, on the CSDN dataset, the unwanted lockout rate is 4.0% for 3-strikes and just 0.5% for CS-all even when no ban-list is used ($B = 0$).

Secondly, we find that increasing the ban-list size B reduces the unwanted lockout rate for DALock. e.g., from 2.56% to 0.08% for 0.1-CS-all after banning 1000 passwords from bfield. Thus, while larger B values might annoy users during the account creation process, they positively impact the lockout rate. For instance, setting $B = 10^5$ makes all DALock implementations achieve 10-strikes level lockout rate, i.e., ≈ 0%. While the unwanted lockout rate for DALock is negatively correlated with B we note that the lockout rate for the traditional K-strikes mechanism is uncorrelated with B since the hit-count is ignored. The lockout rate was approximately 4% (3-strikes) and 0% (10-strikes) for all datasets and ban-list sizes B.

Finally, we found that subsampling minimally affects the usability of CS-based DALock especially when trained on a larger dataset. In fact, when the dataset is small, the usability is often improved. For instance, based on the usability plot of bfield, the unwanted lockout rate of 0.1-CS-5% is 1.25%, which is marginally better than

0.1-CS-all (1.34%). On larger datasets such as csdn and 000webhost this difference becomes negligible (< 0.0001%). To understand why usability improves on smaller datasets we remark that subsampling often causes count sketches to underestimate password frequency (for undersampled passwords) which means that it will often take longer to reach the hit count threshold Ψ. However, for the same reason, subsampling can negatively impact security when the dataset was already small (see section 6.1).

Security When we implement DALock with a differentially private count sketch (ϵ=0.1-CS-all(k,Ψ) or ZXCVBN, we find that the total number of compromised accounts is strictly lower in comparison to the stringent 3-strikes mechanism. This result holds robustly for all datasets and all ban-list sizes. We further remark that (10,Ψ)-DALock will always outperform the traditional 10-strikes mechanism, which is equivalent to (10, ∞)-DALock. As a concrete example, consider the CSDN dataset. When B=0 and the authentication server adopts the 3-strikes mechanism, an attacker compromises approximately 5.8% of user accounts compared with 1.4% when adopting DALock (0.1-CS-all with parameters) or 4.6% when we instantiate with ZXCVBN. As a second concrete example, when we ban the top B=1000 password from bfield, then the attacker compromises 0.536% (resp. 0.08%) of user accounts when adopting the traditional 3-strikes mechanism (resp. DALock with a differentially private count sketch). Recall that the usability of DALock is also vastly superior to our 3-strikes mechanism in this setting.

Secondly, we find that increasing the ban-list size B decreases the percentage of cracked passwords. This result holds whether we adopt DALock or the traditional 3-strikes mechanism though DALock (0.1-CS-ALL) continues to outperform 3-strikes even as the ban-list increases to B=10^5. In fact, we found that DALock with no ban-list (B=0) performs as well as 3-strikes with a larger ban-list of size B=10^4. Thus, increasing B can have a positive usability and security impact though this policy might inconvenience more users during password registration.

Thirdly, we find that 0.1-CS-5% usually performs as well as 0.1-CS-all with an exception for smaller datasets when the ban-list size B is larger. For example, when we train our count sketch on bfield$_{5\%}$, the security of DALock is slightly worse than the traditional 3-strikes mechanism when B >10. This is because we do not have enough data to build an accurate differentially private frequency oracle and the attacker can exploit passwords whose frequencies are underestimated. We also find that other implementations of DALock (e.g., using frequency oracles like Neural Networks or Markov Models) often outperform 3-strikes, but as the ban-list size B grows larger, this is not always the case.

6.3 Summary and Discussion

We find that CS/ZXCVBN-based DALock offers a superior security/usability tradeoff to the classical K-strikes mechanism. DALock can also be reasonably instantiated with password strength models such as Markov Models, Probabilistic Context-Free Grammars, and Neural Networks to achieve a reasonable balance between security and usability. Our simulations also highlight the security *and* usability benefits of banning overly popular passwords given an accurate ban-list. Our analysis shows that the best security/usability tradeoffs can be obtained when the most popular passwords are banned *and* when the DALock frequency oracle is instantiated with a differentially private count sketch or ZXCVBN password strength meter. For large organizations with at least 0.3 million users, we recommend using a ϵ=0.1 differentially private count sketch as the frequency oracle. While for smaller organizations, we recommend implementing DALock with ZXCVBN.

Limitations Our empirical security results are all based on simulations. While we aim to model the authentication server, users, and a powerful attacker, there will inevitably be some differences between the simulated/real-world behavior of the attacker/users. We also remark that our simulations do not model the behavior of targeted attackers. Extending DALock to protect against targeted attackers is an important research question that is beyond the scope of the current paper. Finally, we remark that larger organizations might distribute the workload across multiple authentication servers. In this case maintaining a synchronized state (K_u, Ψ_u) for each user u could be challenging. To address this challenge, it may be necessary to define a relaxation of our DALock mechanism where the states (K_u, Ψ_u) on each authentication server are not always assumed to be perfectly synchronized.

7 CONCLUSION

We present a novel *distribution-aware* password throttling mechanism DALock that penalizes incorrect passwords proportionally to their popularity. We show that DALock can be reliably instantiated with either a password strength model such as ZXCVBN or with a differentially private count sketch. Our empirical simulations demonstrate that DALock offers a superior balance between security and usability and is particularly effective when used combined with a ban-list of overly popular passwords. For example, on the bfield dataset, DALock can reduce the success rate of an attacker to 0.08% whilst simultaneously reducing the unwanted lockout rate to just 0.08% if 1000 passwords are banned. Under the same situation, the classic 3-strikes mechanism results in worse security and usability performance. (0.58% for security and 4.0% for usability).

REFERENCES

[1] Jeremiah Blocki, Manuel Blum, and Anupam Datta. 2013. Naturally Rehearsing Passwords. In *Advances in Cryptology – ASIACRYPT 2013, Part II (Lecture Notes in Computer Science, Vol. 8270)*, Kazue Sako and Palash Sarkar (Eds.). Springer, Heidelberg, Germany, Bengalore, India, 361–380. https://doi.org/10.1007/978-3-642-42045-$0_1$9

[2] Jeremiah Blocki, Anupam Datta, and Joseph Bonneau. 2016. Differentially Private Password Frequency Lists. In *ISOC Network and Distributed System Security Symposium – NDSS 2016*. The Internet Society, San Diego, CA, USA.

[3] Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. 2018. On the Economics of Offline Password Cracking. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 853–871. https://doi.org/10.1109/SP.2018.00009

[4] Jeremiah Blocki, Saranga Komanduri, Ariel Procaccia, and Or Sheffet. 2013. Optimizing password composition policies. In *Proceedings of the fourteenth ACM conference on Electronic commerce*. ACM, 105–122.

[5] Joseph Bonneau. 2012. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 538–552. https://doi.org/10.1109/SP.2012.49

[6] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 553–567. https://doi.org/10.1109/SP.2012.44

[7] Sacha Brostoff and Angela Sasse. 2003. Ten strikes and you're out: Increasing the number of login attempts can improve password usability. (07 2003).

[8] Elie Bursztein, Steven Bethard, Celine Fabry, John C. Mitchell, and Daniel Jurafsky. 2010. How Good Are Humans at Solving CAPTCHAs? A Large Scale Evaluation.

In *2010 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Berkeley/Oakland, CA, USA, 399–413. https://doi.org/10.1109/SP.2010.31

[9] Elie Bursztein, Matthieu Martin, and John C. Mitchell. 2011. Text-based CAPTCHA strengths and weaknesses. In *ACM CCS 2011: 18th Conference on Computer and Communications Security*, Yan Chen, George Danezis, and Vitaly Shmatikov (Eds.). ACM Press, Chicago, Illinois, USA, 125–138. https://doi.org/10.1145/2046707.2046724

[10] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. 2002. Finding Frequent Items in Data Streams. In *ICALP 2002: 29th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science, Vol. 2380)*, Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo (Eds.). Springer, Heidelberg, Germany, Malaga, Spain, 693–703. https://doi.org/10.1007/3-540-45465-9$_5$9

[11] Rahul Chatterjee, Anish Athayle, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. 2016. pASSWORD tYPOS and How to Correct Them Securely. In *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 799–818. https://doi.org/10.1109/SP.2016.53

[12] Rahul Chatterjee, Joanne Woodage, Yuval Pnueli, Anusha Chowdhury, and Thomas Ristenpart. 2017. The TypTop System: Personalized Typo-Tolerant Password Checking. In *ACM CCS 2017: 24th Conference on Computer and Communications Security*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, Dallas, TX, USA, 329–346. https://doi.org/10.1145/3133956.3134000

[13] George B Dantzig. 1957. Discrete-variable extremum problems. *Operations research* 5, 2 (1957), 266–288.

[14] Cynthia Dwork. 2011. Differential privacy. *Encyclopedia of Cryptography and Security* (2011), 338–340.

[15] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *ACM CCS 2014: 21st Conference on Computer and Communications Security*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM Press, Scottsdale, AZ, USA, 1054–1067. https://doi.org/10.1145/2660267.2660348

[16] Dinei Florencio and Cormac Herley. 2007. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 657–666.

[17] David Freeman, Sakshi Jain, Markus Dürmuth, Battista Biggio, and Giorgio Giacinto. 2016. Who Are You? A Statistical Approach to Measuring User Authenticity. In *ISOC Network and Distributed System Security Symposium – NDSS 2016*. The Internet Society, San Diego, CA, USA.

[18] ghacks 2011. Amazon Login May Accept Password Variants. https://www.ghacks.net/2011/01/31/amazon-login-may-accept-password-variants/

[19] Maximilian Golla, Daniel V Bailey, and Markus Dürmuth. 2017. " I want my money back!" Limiting Online Password-Guessing Financially.. In *SOUPS*.

[20] Maximilian Golla and Markus Dürmuth. 2018. On the Accuracy of Password Strength Meters. In *ACM CCS 2018: 25th Conference on Computer and Communications Security*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 1567–1582. https://doi.org/10.1145/3243734.3243769

[21] Ariel Gordon and Richard Allen Lundeen. 2014. Efficiently throttling user authentication. US Patent 8,898,752.

[22] C. Herley and P. Van Oorschot. 2012. A Research Agenda Acknowledging the Persistence of Passwords. *IEEE Security Privacy* 10, 1 (Jan 2012), 28–36. https://doi.org/10.1109/MSP.2011.150

[23] Dmitry Kogan, Nathan Manohar, and Dan Boneh. 2017. T/Key: Second-Factor Authentication From Secure Hash Chains. In *ACM CCS 2017: 24th Conference on Computer and Communications Security*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, Dallas, TX, USA, 983–999. https://doi.org/10.1145/3133956.3133989

[24] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. 2011. Of passwords and people: measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2595–2604.

[25] Ariel Kulik and Hadas Shachnai. 2010. There is no EPTAS for two-dimensional knapsack. *Inform. Process. Lett.* 110, 16 (2010), 707–710.

[26] David Malone and Kevin Maher. 2012. Investigating the distribution of password choices. In *Proceedings of the 21st international conference on World Wide Web*. ACM, 301–310.

[27] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2016. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In *USENIX Security 2016: 25th USENIX Security Symposium*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, Austin, TX, USA, 175–191.

[28] Moni Naor, Benny Pinkas, and Eyal Ronen. 2019. How to (not) Share a Password: Privacy Preserving Protocols for Finding Heavy Hitters with Adversarial Behavior. In *ACM CCS 2019: 26th Conference on Computer and Communications Security*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 1369–1386. https://doi.org/10.1145/3319535.3363204

[29] Benny Pinkas and Tomas Sander. 2002. Securing Passwords Against Dictionary Attacks. In *ACM CCS 2002: 9th Conference on Computer and Communications Security*, Vijayalakshmi Atluri (Ed.). ACM Press, Washington, DC, USA, 161–170. https://doi.org/10.1145/586110.586133

[30] prowebscraper. 2019. Top 10 Captcha Solving Services Compared. https://prowebscraper.com/blog/top-10-captcha-solving-services-compared/

[31] RockYou 2010. RockYou Password Corpus. http://downloads.skullsecurity.org/passwords/rockyou.txt.bz2.

[32] Ravi Sandhu, Colin Desa, and Karuna Ganesan. 2005. System and method for password throttling. US Patent 6,883,095.

[33] Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. 2010. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX conference on Hot topics in security*. USENIX Association, 1–8.

[34] Stuart Schechter, Yuan Tian, and Cormac Herley. 2019. StopGuessing: Using guessed passwords to thwart online guessing. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 576–589.

[35] S Schecter and C Herley. 2016. The Binomial Ladder Frequency Filter and its Applications to Shared Secrets. *MSR-TR-2018-18* (2016).

[36] Apple Differential Privacy Team. [n.d.]. Learning with Privacy at Scale. https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html Retrieved 25, Apr. 2019.

[37] TechNewsWorld 2019. Microsoft Exposes Russian Cyberattacks on Phones, Printers, Video Decoders. https://www.technewsworld.com/story/86171.html

[38] Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. 2015. Measuring Real-World Accuracies and Biases in Modeling Password Guessability. In *USENIX Security 2015: 24th USENIX Security Symposium*, Jaeyeon Jung and Thorsten Holz (Eds.). USENIX Association, Washington, DC, USA, 463–481.

[39] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. 2003. CAPTCHA: Using Hard AI Problems for Security. In *Advances in Cryptology – EUROCRYPT 2003 (Lecture Notes in Computer Science, Vol. 2656)*, Eli Biham (Ed.). Springer, Heidelberg, Germany, Warsaw, Poland, 294–311. https://doi.org/10.1007/3-540-39200-9$_1$8

[40] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. 2008. recaptcha: Human-based character recognition via web security measures. *Science* 321, 5895 (2008), 1465–1468.

[41] Ding Wang, Haibo Cheng, Ping Wang, Xinyi Huang, and Gaopeng Jian. 2017. Zipf's law in passwords. *IEEE Transactions on Information Forensics and Security* 12, 11 (2017), 2776–2791.

[42] Ding Wang, Gaopeng Jian, Xinyi Huang, and Ping Wang. 2014. Zipf's Law in Passwords. Cryptology ePrint Archive, Report 2014/631. http://eprint.iacr.org/2014/631.

[43] Ding Wang and Ping Wang. 2016. On the Implications of Zipf's Law in Passwords. In *ESORICS 2016: 21st European Symposium on Research in Computer Security, Part I (Lecture Notes in Computer Science, Vol. 9878)*, Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows (Eds.). Springer, Heidelberg, Germany, Heraklion, Greece, 111–131. https://doi.org/10.1007/978-3-319-45744-4$_6$

[44] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. 2016. Targeted Online Password Guessing: An Underestimated Threat. In *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, Vienna, Austria, 1242–1254. https://doi.org/10.1145/2976749.2978339

[45] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally Differentially Private Protocols for Frequency Estimation. In *USENIX Security 2017: 26th USENIX Security Symposium*, Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, Vancouver, BC, Canada, 729–745.

[46] Daniel Lowe Wheeler. 2016. zxcvbn: Low-Budget Password Strength Estimation. In *USENIX Security 2016: 25th USENIX Security Symposium*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, Austin, TX, USA, 157–173.

[47] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. 2018. Yet Another Text Captcha Solver: A Generative Adversarial Network Based Approach. In *ACM CCS 2018: 25th Conference on Computer and Communications Security*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 332–348. https://doi.org/10.1145/3243734.3243754

Notation	Description
(K, Ψ)-DALock	DALock with strike threshold K and hit count threshold Ψ
$\mathcal{A}$	$\mathcal{A}$dversary
$\mathcal{U}$	A set of $\mathcal{U}$sers
u	A user $u \in \mathcal{U}$
$\mathcal{P}$	The set of all potential user $\mathcal{P}$asswords
$\mathcal{D}_{\mathcal{U}} \subseteq \mathcal{P}$	a multiset of N sampled passwords for users $u_1, \ldots, u_N \in \mathcal{U}$
pw_u	User u's password
pw_r	The r'th most likely password in $\mathcal{D}_{\mathcal{U}} \subseteq \mathcal{P}$
CS	Count (Median) Sketch data structure
$\mathsf{F}(pw, \mathcal{D}_{\mathcal{U}})$	Frequency of password pw in dataset $\mathcal{D}_{\mathcal{U}}$
$\mathsf{P}(pw)$	Empirical probability of password pw
Estimate(pw)	Estimated frequency of password pw
$\mathsf{p}(pw)$	Estimated probability of password pw
Ψ	Hit count threshold
Ψ_u	Cumulative hit count threshold on u's account. The account gets locked out if Ψ_u exceeds Ψ
K	Traditional strike threshold.
K_u	Cumulative strike threshold on u's account. The account gets locked if K_u exceeds K.

Table 2: Notation Summary

A PASSWORD KNAPSACK IS NP HARD

In this section, we supplement the details on proving PK is NP hard by showing a reduction from a well-known NPhard problem subset-sum to it. We begin this by first formally define the subset sum problem and then prove password knapsack is NP hard by showing the reduction from subset sum to it.

Definition 2 (Subset Sum). *Given partition instance $x_1, \ldots, x_n \in (0, 2^m]$ and target sum value T. The goal is to find $S \subseteq [n]$ s.t. $\sum_{i \in S} x_i = T$.*

Theorem A.1 (Hardness of Password Knapsack). *Find optimal solutions for password knapsack is NP-hard.*

Proof: Reduction: One can create the following password knapsack instance

- Set $\gamma = \sum_{i=1}^{n} x_i$,
- Set $\psi = T/(2\gamma) < \frac{1}{2}$,
- Set $CS(p_i) = f(p_i) = x_i/(2\gamma)$ for $i = 1, \ldots, n$
- Set $f(p_{last}) = 1 - \sum_{i=1}^{n} p_i = 1/2 > \psi$.

If S exists for partition instance, then the attacker can use S for password knapsack to crack $p_{last} + T/(2\gamma)$ passwords. On the other hand, let S be the optimal password knapsack solution such that $\sum_{i \in S} CS(p_i) \leq \psi$, then the attacker cracks at most $p_{last} + \sum_{i \in S} f(p_i) \leq 1/2+\psi$ passwords. If equality holds, then $\sum_{i \in S} f(p_i) = \psi$ which implies $\sum_{i \in S} x_i = T$ by definition of ψ.

B SOLVING PK WITH HEURISTIC

In this section, we discuss two heuristic approaches, DAB and FMPPF, described in **Section** 5.4.

The DAB approach takes three inputs: a sorted password dictionary $\mathcal{P}_{\tilde{\Pi}} = \{p\tilde{w}_1, \ldots, p\tilde{w}_n\}$, hit count budget Ψ and strike count budget K. $\mathcal{P}_{\tilde{\Pi}}$ is sorted based on the ratio of actual popularity and estimated popularity, i.e., $\frac{\mathsf{p}(pw)}{\mathsf{P}(pw)}$: $\mathcal{P}_{\tilde{\Pi}} = \{pw_{\tilde{\Pi}(1)}, \ldots, pw_{\tilde{\Pi}(n)}\}$. The algorithm keeps placing passwords into the knapsack S based on the sorted order until it cannot further add some password pw, i.e., $\mathsf{p}(S \cup pw) \geq \Psi$. At this point, DAB compares $\mathsf{P}(pw)$ with $\mathsf{P}(S)$ and sets S to be the one with the higher value. The above process is repeated until the whole dictionary is scanned. In the end, the algorithm returns K passwords based on their actual probabilities.

Primary incentives of using DAB are 1) to take advantage of underestimated passwords and 2) to avoid (severely) overestimated ones. However, there are several drawbacks of DAB. Firstly, the progress can be slow because priorities are given to significantly underestimated passwords, i.e., rare passwords. Intuitively, the ratio $\frac{\mathsf{P}(pw)}{\mathsf{p}(pw)}$ of popular password pw, i.e., $\mathsf{P}(pw)$ is large, is likely to be close to 1; therefore, attempts with popular ones are likely to be delayed. Secondly, unlike the vanilla version of Knapsack, DAB may not yield a 2-approximation due to the additional constraint on the number of passwords. Third, the computation cost of DAB is high because the algorithm has to go through the whole dictionary (for each run). Consider $\mathcal{A}$ usually attack multiple accounts simultaneously, DAB may not be the heuristic to be used.

A faster alternative to DAB is FMPPF. It takes three input parameters: password dictionary $\mathcal{P} = \{pw_1, \ldots, pw_n\}$, hit count budget Ψ, and strike budget K. FMPPF selects passwords greedily as well but using different criteria. $\mathcal{P}$ is a password dictionary sorted based on the actual popularity only. In addition, to save computational cost, FMPPF terminates once it finds K passwords suitable for attacks and stops further exploring the dictionary.

In short-time attack scenarios, FMPPF offers a better chance of success than DAB by attempting popular ones first. For long-term attacks, FMPPF should still be able to achieve almost optimal results given an abundant choice of passwords. In fact, based on the empirical results (in **section** 6.1), the performance of FMPPF is very close to theoretical upper bounds $(\Psi + \mathsf{P}(pw_1))$.

C SIMULATING USERS' MISTAKES

In this section, we elaborate on the missing details for simulating users' mistakes. To help readers visualize the process, we plot a flowchart in **figure** 6. The starting point is to simulate the recall error. Following existing works [11, 12], we set the probability of making a recall error to be 2.4%. When making a recall error, we assume that each user will choose one of their five "passwords from other services" uniformly, i.e., w.p. 20%. After this step, we further simulate typos (on the password intended to enter) w.p. approx. 5%. Condition on making typos, we simulate this step by choosing a typo type with their conditional probability based on existing works[11, 12](e.g., insert an extra letter w.p. 12%.). Notice that a user can make both mistakes. e.g., recall the wrong password pw' and typo pw'.

D MORE EXPERIMENTAL RESULTS

In this section, we provide more detailed experimental results for readers to understand the underlying details of DALock. We elaborate on each frequency oracle's security and usability performance with wider Ψ range: $\{2^{-8}, 2^{-9}, 2^{-10}, 2^{-11}, 2^{-12}\}$. For count sketch implementation, we show extra results on applying subsampling and differential privacy with the following testing parameters:[7]

[7] Detailed experimental plots are available in the full version of the paper. We selected several plots that can show the characteristics of various implementations of DALock

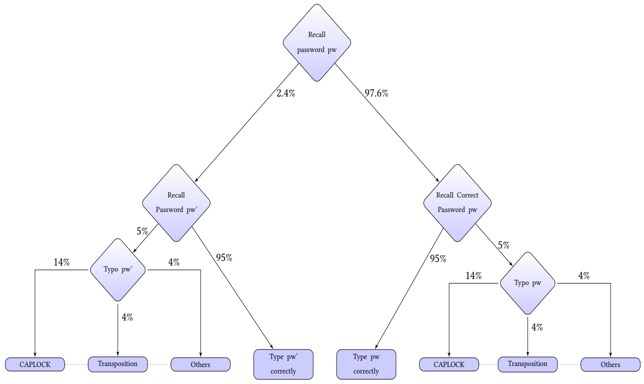

Figure 6: Flow Chart for Simulating Users' mistake

- Subsampling rate: 1%, 5%, 10%, 100%(all)
- Differential privacy budget: 0.1, 0.5, 1.0, ∞

We begin by discussing the pros and cons of each frequency oracle based on our results. And then provide our insights on how to deploy DALock with it. Finally, we make our overall recommendation by comparing the performance of them.

PCFG/NeuralNet/Markov/HashCat/Min As discussed previously, those models use "guessing number" to indicate the strength of passwords. Based on our observation, they have their best security/usability advantages (compared to the 3-stikes mechanism) when high Ψ is used and ban-list size $|B|$ is small. We notice that adopting larger Ψ hardly impacts security performance despite the fact that usability can be benefited. In fact, We only observe noticeable security impact on the Hashcat model based on brazzers, clixsense, and rockyou dataset.

All those frequency oracles can be used to implement DALock to achieve better security than the 3-strikes mechanism; however, they gradually lose the security advantages as the ban-list size increases, e.g., banning 100 passwords results in worse security compared to the 3-strikes mechanism on all datasets. For usability, all five implementations can be configured to have lower lockout rates than the 3-strikes mechanism. e.g., using $\Psi \geq 2^{-9}$ results in strictly better usability than the 3-strikes mechanism across all datasets for all models.

Based on our observation, deploying DALock with those five frequency oracles is not recommended if the server can accurately identify and ban approx. 100 popular passwords. In addition, larger Ψ is recommended, e.g., using $\Psi = 2^{-9}$ to achieve strictly better security/usability performance compared to the 3-strikes mechanism.

ZXCVBN To achieve the optimal security/usability trade-off, we recommend deploying ZXCVBN with $\Psi = 2^{-9}$ combined with a large ban-list. Unlike the previously mentioned five estimators, Ψ impacts both security and usability performance sensitively. On the positive side, one can also sharpen *both* the security and usability by adopting larger ban-list, e.g., B = 1000.

DALock can be easily implemented by ZXCVBN to achieve strictly better security *and* usability compared to the 3-strikes mechanism. Our results show that adopting any $\Psi \leq 2^{-8}$ results in security advantage (compared to the 3-strikes mechanism) across all datasets even with a large ban-list; however, we do observe that ZXCVBN overestimate many rare passwords. Thus, it's crucial to adopt $\Psi \geq 2^{-9}$ for usability practice. Combining the security and usability results, we conclude that using $\Psi = 2^{-9}$ yields the optimal security/usability trade-off. i.e. ZXCVBN(K:10,Ψ:2^{-9}) is more secure than the 3-strikes mechanism and has approx. 0% lockout rate.

In conclusion, deploying DALock with ZXCVBN is recommended when it is hard to obtain accurate password distribution description. Based on the empirical results, setting $\Psi = 2^{-9}$ *and* banning popular passwords yields the best security/usability trade-off.

Differentially Private Count Sketch In this section, we focus discussion on the impact of the following three parameters: hitcount Ψ, sampling rate r, and privacy budget ϵ. The experimental plots are available in the full version of the paper.

Tunning Ψ for optimal security/usability trade-off on a differentially private Count Sketch is a less challenging task compared to other frequency oracles. Our results show that 0.1-CS-all can achieve strictly better security and usability than the 3-strikes mechanism for $\Psi \in [2^{-8}, 2^{-10}]$ on all datasets and with all ban-list sizes. In addition, we observe that 0.1-CS-all reaches approx. 0% lockout rate if 100 or more passwords are banned when $\Psi \in [2^{-8}, 2^{-10}]$.

To investigate how many users one needs to accurately build a differentially private count sketch, we train count sketches with subsampled datasets - $\mathcal{D}_{\mathcal{U}_{1\%}}$, $\mathcal{D}_{\mathcal{U}_{5\%}}$, and $\mathcal{D}_{\mathcal{U}_{10\%}}$ - in addition to $\mathcal{D}_{\mathcal{U}}$. Our simulation results show that lower sampling rates can hurt security as $\mathcal{A}$ can take advantage of underestimated passwords. We also observe that 0.1-CS-10%/0.1-CS-5%/0.1-CS-1% can be just as accurate as 0.1-CS-all when we have more than 2/6/32 millions users in the $\mathcal{D}_{\mathcal{U}}$(see clixsense/csdn/RockYou). This result empirically shows organizations need approx. 0.2-0.3 million users to train an *accurate* differentially private Count Sketch.

To study how privacy noise can perturb security/usability performance of well-tuned differentially privacy Count-Sketch (e.g., with throttling parameters k = 10 and $\Psi = 2^{-10}$) in bad scenarios, we experiment training Count Sketch on small datasets (e.g., $\mathcal{D}_{\mathcal{U}_{1\%}}$) with practically small privacy budgets. Our results demonstrate the security/usability performance of three different differentially Count-Sketches: 0.1-CS-1%, 0.5-CS-1%, and 1.0-CS-1%. In addition, we observe that even 0.1-differential privacy had minimal impact on both security and usability performance of Count Sketches.

In brief, the empirical results show that differentially private Count Sketch can be easily trained with low privacy budget cost, e.g., $\epsilon = 0.1$ and with as few as 0.2-0.3 million users. It's also the easiest frequency oracle to tune for security/usability performance. We recommend large entities to deploy DALock with differentially private Count Sketch once the above criteria can be met.

Deploying DALock we found two feasible solutions to instantiate (10, Ψ)-DALock based on experimental results - differentially private count sketches and ZXCVBN password strength meter. We recommend deploying DALock with a 0.1-differentially private count sketch with $\Psi \in [2^{-8}, 2^{-10}]$ when the authentication server can collect at least 0.3 million passwords. Otherwise we recommend using ZXCVBN($K : 10, \Psi : 2^{-9}$) to instantiate DALock. Banning popular passwords is recommended for both apporaches to achieve better security/usability results.

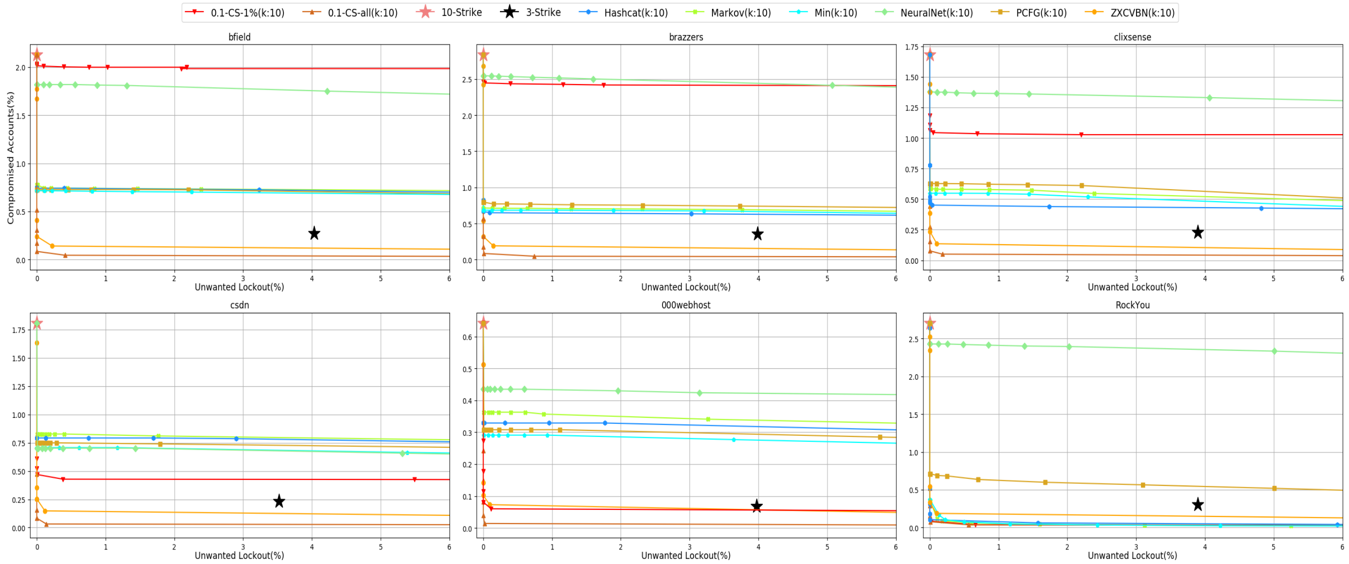

Figure 7: Usability/Security Trade-off(Banlist Size = 1000)

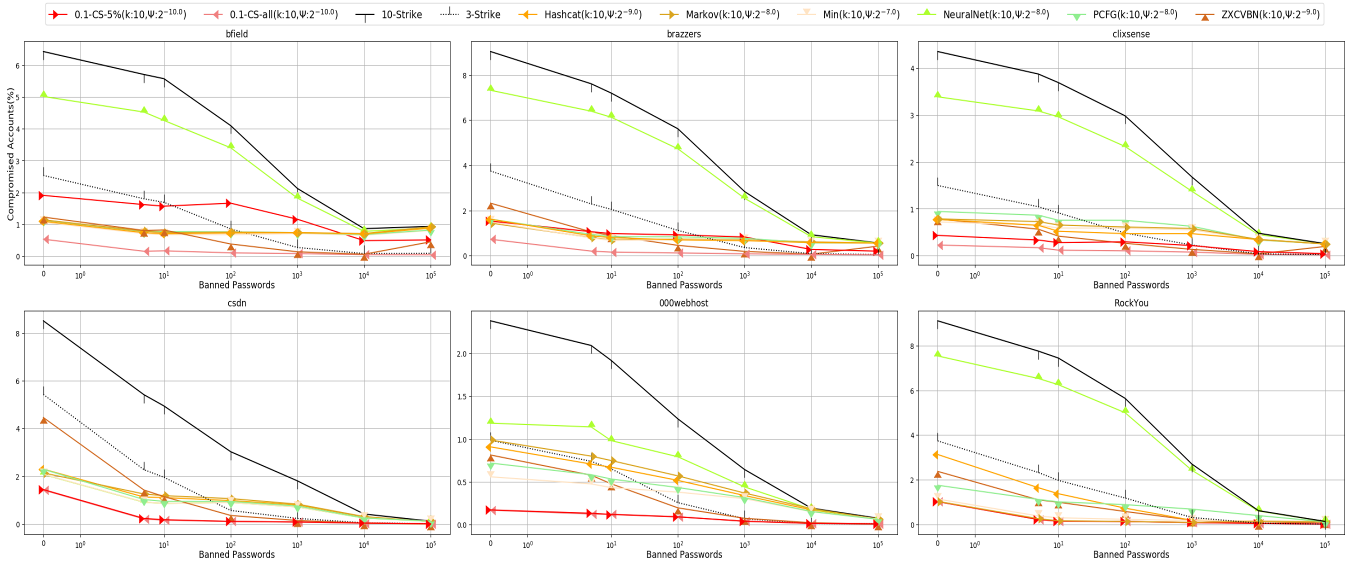

Figure 8: Security Measurement of DALock (All Datasets)

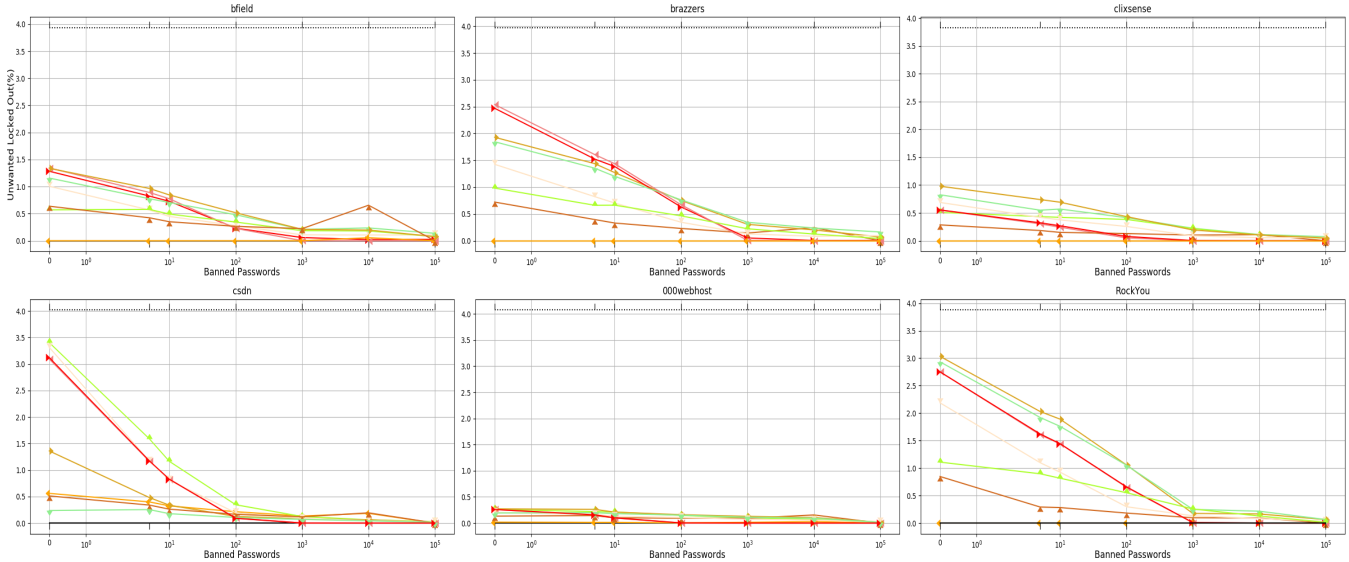

Figure 9: Usability Measurement of DALock(All Datasets)