DALock: Password Distribution-Aware Throttling

Abstract: Large-scale online password guessing attacks are widespread and pose a persistent privacy and security threat to users. 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, our simulations indicate that DALock offers a superior simulated 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).

Keywords: Authentication Throttling; Password; Dictionary Attack

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. Online attacks pose a significant risk to user privacy and security as the human tendency to pick weak (“low-entropy”) passwords has been well documented, e.g., see [6]. 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 [45] might additionally incorporate background knowledge about the specific target user (e.g., birthday, 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. Brostoff et al. [8] advocated for a larger value of $K$ (e.g., $K = 10$) to reduce the unwanted lockout rate but this may increase the vulnerability to online attacks. Renaud et al. [33] suggested the intermediate threshold $K = 5$ to achieve a better balance between security and usability.

Bonneau et al. [7] 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 [24]. Thus, for the foreseeable future, protecting passwords against online attacks without locking out legitimate users remains a crucial challenge for usable privacy and security.
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 [26]. 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 [19, 34] to help distinguish between malicious and benign login attempts. While these features can be helpful indicators, they are not failproof and their usage can raise privacy concerns. Honest users will sometimes login from different devices, different geographical locations (e.g., during travel) and at times which can occasionally deviate from their normal schedule [19]. 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” $\Psi_u$ for each user $u$, where $\Psi_u$ intuitively represents the cumulative probability mass of all incorrect login attempts for user $u$’s account. When $\Psi_u$ exceeds the hit count threshold $\Psi$, we decide to lock the account.

**Example 1: Usability** Figure 1 describes an example scenario where a user, who would have been locked out under the standard 3-strikes mechanism, is able to successfully authenticate with DALock. In this example scenario, our user John Smith registers an account with the somewhat complicated password “J.S.UsesStrongpwd!” 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 we will not reach the hit count threshold and DALock would allow our user to continue attempting to login until he remembers the correct password.

**Example 2: Security/Privacy** Figure 1 also 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 strength of each incorrect login attempt to update $\Psi_u$. We propose two implementations: password strength models (e.g., ZXCVBN [47]) 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 the 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 pass-
words to check which maximizes his success rate subject to the constraint that the total estimated hit count does not exceed the threshold $\Psi_u$. 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 our model of honest user login attempts/mistakes is informed by prior empirical studies of password typos [12, 13]. Our simulations show that when the hit count threshold $\Psi$ is tuned appropriately, DALock significantly outperforms $K$-strikes mechanisms. In particular, when user accounts are under attack in our simulation, 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 find that the unwanted lockout rate for DALock is much lower compared to $K=3$ strikes mechanism when user accounts are not under attack in our simulation. The simulated usability\(^1\) 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 $\Psi_u$ 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.

\(^1\)Our usability evaluation is based entirely on simulated user behavior. To avoid cumbersome notation we will write typically write “usability” instead “simulated usability” in the remainder of the paper. We acknowledge that one would need to conduct a longitudinal user study to rigorously evaluate the usability of DALock in practice.

2 Related Work and Background

Feature-Based Throttling Mechanism Modern throttling mechanisms [23, 34] 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 [19]. DALock takes an orthogonal approach and relies instead on the popularity of the password guesses instead of potentially confidential user profiles. 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. [36] developed an IP-based throttling mechanism called StopGuessing 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 [37] 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 $\Psi_u$ 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.
Password Distribution  To analyze online statistical guessing attacks it is important to understand the distribution of user passwords. Password distributions have been extensively studied since the last decades [6, 18, 28]. Wang et al. [42–44] observed that Zipf's law distributions nicely fit leaked password corpora and Blocki et al. [4] later found that the same for the differentially private Yahoo! password frequency corpus [3, 6]. Other work has used password cracking models and/or statistical techniques to characterize the password distribution.

Password Typos  Recent studies [12, 13] 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 authentication mechanisms and demonstrate that typo-corrective does not come at the cost of security — similar mechanisms have already been deployed in the industry [20]. In our usability simulations for DALock, we leverage the findings of [12, 13] to help simulate honest user mistakes during authentication.

Increasing Cost of Authentication  Pinkas and Sanders [31] 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 [40]. For example, reCAPTCHA [41] 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 [9, 10]. Nevertheless, an attacker can always pay humans to solve CAPTCHA challenges [32]. Besides, sophisticated CAPTCHA solvers [48] powered by neural networks make it increasingly challenging to design CAPTCHA puzzles that are also easy for a human to solve. Golla et al. [21] 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 [5, 26]. An alternate approach of Schechter et al. [35] 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. [5] shows that this is the optimal approach to boost the minimum entropy of the password distribution.

3 Preliminaries

3.1 Notation Summary

In this section, we summarize frequently used notations in this paper across all sections in Table 3, Appendix.

We use \( \mathcal{U} = \{u_1, \ldots, u_N\} \) to denote a set of \( N \) users and \( \mathcal{D}_u = \{pw_{u_1}, \ldots, pw_{u_N}\} \) to denote the corresponding multiset of user passwords i.e., \( pw_u \in \mathcal{P} \) denotes the password selected by user \( u \in \mathcal{U} \). We typically view \( \mathcal{D}_u \) as \( N \) independent samples from an underlying distribution over \( \mathcal{P} \) and use \( P(pw) \) to denote the probability that a user selects 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 \( P(pw_1) \geq P(pw_2) \ldots \). We use \( F(pw, D_u) = \{|i : pw_u = pw|\} \) to denote the number of times the password \( pw \) was observed in our sample — when the dataset is clear from context we will sometime drop \( D_u \) and simply write \( F(pw) \).

We remark that \( P(pw) = \frac{1}{N} \sum_{i=1}^{N} F(pw, D_u) \) and thus for popular passwords we expect that the estimate \( \hat{P}(pw) \approx \frac{F(pw, D_u)}{N} \) will be accurate as long as our sample size \( N \) is sufficiently large. However, because the underlying password distribution is unknown and an authentication server cannot store a plaintext encoding of \( D_u \) we will often use other techniques to estimate \( P(pw) \) and/or \( F(pw, D_u) \). In particular, we consider a count sketch data structure \( \text{CS} \) trained on \( D_u \) (or a small subsample of \( D_u \)), which allows us to generate an estimate \( \hat{p}(pw) \) for the true probability \( P(pw) \) of each password \( pw \). Similarly, we can also use password strength meters to compute an estimate \( \hat{p}(pw) \) for \( P(pw) \).

3.2 Count Sketch

The count sketch [11] is a succinct data structure which allows for one to quickly obtain an approximation of the frequency of any item in a dataset. Intuitively, the count sketch data structure supports four operations: Initialize, Add, Estimate and TotalFreq. The operation Add takes as input an item \( x \) (password) and updates the internal state \( \sigma \) of the count-sketch. Similarly, the Estimate operation takes as input an item \( x \) (password) and outputs an estimate of the number of times that this
particular item \( x \) has been added to the count sketch and TotalFreq outputs the total number of items added to the count-sketch. In our implementation, the state \( \sigma : \mathbb{R}^{d \times w} \times \mathbb{R} \) of a count sketch (CS) is represented by a two-dimensional \( d \times w \) array \( \text{CS.ARRAY} \) where \( d \) (depth) and \( w \) (width) are parameters of the count-sketch which can be tuned to balance accuracy and space usage, and a total frequency counter \( \text{CS.T} \). More formally the API for a count sketch can be defined as follows:

\[
\sigma_0 \leftarrow \text{Initialize}(d, w) \quad \text{This function takes as input the count sketch parameters depth/width parameters } d \text{ and } w \text{ and outputs an initial state } \sigma_0 = \emptyset^{d \times w} \times 0, \text{ i.e., an all-zero table. Intuitively, we expect that } \text{TotalFreq}(\sigma_0) = 0 \text{ and that } \text{Estimate}(pw, \sigma_0) = 0 \text{ for each item/password } pw \text{ since no items have been added yet.}
\]

\[
\sigma_{\text{new}} \leftarrow \text{Add}(pw, \sigma) \quad \text{This function takes as input the current state } \sigma \text{ and an item/password } pw \text{ to add and updates the state of the count sketch to } \sigma_{\text{new}}. \text{ Intuitively, we expect that } \text{TotalFreq}(\sigma_{\text{new}}) = \text{TotalFreq}(\sigma) + 1 \text{ and that } \text{Estimate}(pw, \sigma_{\text{new}}) = \text{Estimate}(pw, \sigma) + 1 \text{ i.e., the total count and the count for } pw \text{ are incremented by 1. Because the data-structure is succinct it is possible that the operation slightly interferes with the estimates for other items/passwords } pw' \neq pw \text{ besides the one we are adding i.e., we may have } \text{Estimate}(pw', \sigma_{\text{new}}) \neq \text{Estimate}(pw', \sigma). \text{ For our purposes we do not need to describe the precise details of how the state } \sigma \text{ is updated. However, we remark that in our count median sketch implementation the L1 distance between } \sigma \text{ and } \sigma_{\text{new}} \text{ is upper bounded by } ||\sigma - \sigma_{\text{new}}||_1 \leq d + 1 \text{ — this observation will be used later to tune noise levels for differential privacy. Given a multiset } D_U = \{pw_1, \ldots, pw_d\}, \text{ we use } \sigma_{\text{DU}} = \text{Add}(D_U, \sigma_0) = \text{Add}(pw_1, \text{Add}(pw_2, \text{Add}(pw_3, \ldots, \text{Add}(pw_N, \sigma_0)))) \text{ to denote the final state of the count sketch after all passwords in the dataset } D_U \text{ have been added. When the context is clear we also omit the subscript } D_U \text{ and simply use } \sigma \text{ to denote } \sigma_{\text{DU}}.
\]

\[
\text{Estimate}(pw, \sigma) : \text{ This function takes as input an item/password } pw \text{ and the current count sketch state } \sigma \text{ and outputs an estimate for the frequency of } pw \text{ without updating the count sketch state } \sigma. \text{ Intuitively, we want the estimator to have the following correctness property: } \text{Estimate}(pw, \sigma) = F(pw, D_U), \text{ where } F(pw, D_U) \text{ denotes the actual frequency of } pw \text{ in } D_U.
\]

\[
\text{TotalFreq}(\sigma) : \text{ this operation takes as input the current count sketch state } \sigma \text{ and outputs the total number of items that have been added to the count sketch e.g., if } \sigma_0 = \text{Initialize}(D, w) \text{ and } \sigma_{D_u} = \text{Add}(D_u, \sigma_0) \text{ then } \text{TotalFreq}(\sigma) \text{ returns } |D_u|. \text{ The state } \sigma \text{ is not updated.}
\]

We denote the estimated popularity of a password \( pw \) by \( \sigma \) with \( p(pw, \sigma) = \frac{\text{Estimate}(pw, \sigma)}{\text{TotalFreq}(\sigma)} \). For the rest of the discussion, we sometimes omit \( \sigma \) when there is no ambiguity to simplify the presentation. e.g. \( p(pw) = p(pw, \sigma) \).

In addition, we allow the above APIs to take a set of passwords as an argument and return the summed results. i.e. \( p(S) = \sum_{pw \in S} p(S) \).

We elect to use the count (median) sketch [11] data structure in this work as it is invariant to the order in which passwords are added (e.g., \( \text{Add}(\{pw_1, \ldots, pw_d\}, \sigma) = \text{Add}(\{pw_N, \ldots, pw_1\}, \sigma) \)), and because it can easily be modified to preserve differential privacy. StopGuessing[36], used an alternative data-structure called 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 binomial ladder is not suitable for DADLock for two reasons. First, DADLock requires a fine-grained estimate of each password’s popularity while a binomial ladder was designed to provide a binary classification i.e., either the password is a “heavy hitter” or it is not. Second, the binomial ladder is not invariant to the order in which passwords are added e.g., it can overestimate the frequency of recently popular passwords.

### 3.3 Differential Privacy

While the succinct count-sketch data structure is a useful tool to approximate the frequency of a particular password in the dataset, its usage raises a natural privacy concern. Could the attacker infer anything about a particular user’s password from the count-sketch \( \sigma \) if the authentication server was breached? We address these concerns by using a differentially private count sketch. Differential privacy [15] is a compelling mathematical definition of privacy that has begun to see industrial deployment[17]. 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. In our password context we can define differential privacy as follows.

**Definition 1 (\( \epsilon \)-Differential Privacy [15]):** A randomized mechanism \( M \) gives \( \epsilon \)-differential privacy if for any pair of neighboring datasets \( D_U \) and \( D'_U \), and any \( \sigma \in \text{Range}(M) \), \( \Pr[M(D_U) = \sigma] \leq e^{\epsilon} \cdot \Pr[M(D'_U) = \sigma] \).
We consider two datasets $D_u$ and $D'_u$ to be neighbors if either $D_u = D'_u + pw_u$ or $D'_u = D_u + pw_u$, where $D_u + pw_u$ denotes the dataset resulted from adding the password $pw_u$ to the dataset $D_u$. We use $D_u \simeq D'_u$ to denote two neighboring datasets. Differential privacy protects the privacy of any individual password in the dataset $D_u$ because adding or removing any single password results in $e^\epsilon$-multiplicative-bounded changes in the probability distribution of the output. If an adversary can make a certain inference about a password based on the output, then the same inference is also likely to occur even if the password does not appear in the dataset.

**Laplace Mechanism** The Laplace mechanism is a classic tool to achieve differential privacy i.e., given any function $f(x) \in \mathbb{R}^{wd+1}$ the mechanism $M(x) = f(x) + (Z_1, \ldots, Z_{wd+1})$ is $e$-differentially private where for each $i \leq wd + 1$ the random variable $Z_i$ is sampled from the Laplace Distribution with PDF $\frac{e^{-|Z_i|}}{2\epsilon}$.

Here, $GS_f$ denotes the global sensitivity of the function $f$ and the noise distribution also depends on the privacy parameter $\epsilon$. In our particular case the global sensitivity of the function $f(D_u) = \text{Add}(D_u, \text{Initialize}(d, w))$ is $GS_f \leq d + 1$ i.e., given any two neighboring datasets $D_u$ and $D'_u$, we have $\|f(D_u) - f(D'_u)\|_{1} \leq d + 1$. Formally, we use $\sigma_{dp} \leftarrow \text{DP}(\epsilon, \sigma)$ to denote a function which (1) samples laplace noise $(Z_1, \ldots, Z_{wd+1})$ according to the PDF $\frac{e^{\frac{-|Z_i|}{\epsilon}}}{2\epsilon}$, and (2) outputs $\sigma_{dp} = \sigma + (Z_1, \ldots, Z_{wd+1})$ to obtain a $\epsilon$-differentially private count sketch state $\sigma_{dp}$. The noise can be added during initialization i.e., we can equivalently set $\sigma_0 = (Z_1, \ldots, Z_{dw+1})$ instead of $\sigma_0 = (0, \ldots, 0)$ during initialization and then compute the final count-sketch state as $\sigma = \text{Add}(D_u, \sigma_0)$.

**Differentially Private Count Sketch: Threat Model** In our threat model we consider an adversary who obtained a snapshot of the count-sketch state e.g., $\sigma = \text{Add}(D_u, \text{Initialize}(d, w)) + (Z_1, \ldots, Z_{wd+1})$. Intuitively, differential privacy ensures that the attacker will not be able to use the snapshot $\sigma$ to draw inferences about any individual password $pw_u$. However, we do not provide privacy guarantees against an attacker who can continuously monitor the state of the count sketch as passwords are added over time e.g., if the attacker learns the initial state $\sigma_0 = (Z_1, \ldots, Z_{wd+1})$ as well as the final state $\sigma = \text{Add}(D_u, \sigma_0)$ then the attacker can easily compute $\sigma - \sigma_0 = \text{Add}(D_u, \text{Initialize}(d, w))$ to remove the noise that we added to preserve differential privacy. We could adopt a stronger privacy notion such as pan-privacy [16] to protect against an attacker who can obtain multiple snapshots of the count sketch state. However, we note that an attacker who is continuously present on the authentication server would (most likely) also be able to observe the plaintext passwords directly. Thus, the practical privacy benefits of using a pan-private count sketch may not be significant.

**Differential Privacy in Passwords** Naor et al.[30] designed a locally differentially private mechanism to identify the most popular passwords in a distribution. Blocki et al. [3] developed a differentially private mechanism for integer partitions and used this to release a private summary of the Yahoo! password dataset.

### 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 bases lockout decisions on the popularity of incorrect password guesses so that an online attacker attempting to use popular passwords will be locked out more quickly while honest typos are punished less severely.

#### 4.1 DALock

**Recap: K-Strikes Mechanism** As briefly discussed in the introduction 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. $K_u$ is incremented by 1 upon each failed login attempt and reset to $K_u = 0$ upon a successful login. Whenever we exceed the threshold $K_u \geq K$ the throttling mechanism kicks in and the authentication server locks the account until the user $u$ takes corrective action e.g., reset password by phone/e-mail or solve a CAPTCHA challenge.

**Extending the K-Strikes Mechanism** The key idea behind DALock is to additionally maintain an extra “hit count” variable $\Psi_u$ for each user $u$. Intuitively, $\Psi_u$ measures the total probability mass of all incorrect guesses submitted on $u$’s account. Initially, when a new user $u$ registers, we will have $\Psi_u = 0$ (and $K_u = 0$). After each failed attempt with an incorrect password $pw \neq pw_u$, the hit count variable $\Psi_u$ and strike count variable $K_u$ will be increased by $p(pw)$ and 1, respectively. i.e., $\Psi_u := p(pw)$, and $K_u := 1$. Here, $p(pw)$ denotes (an estimate of) the probability of the password $pw$. For example, suppose that the (estimated)
probability of the passwords “aaa,” “bbb,” and “ccc” were 3%, 1.7% and 0.8%, respectively. If a user registers with password “ddl” and then attempts to login with the previous three passwords, $\Psi_u$ will be set to $0.055 = 0.03 + 0.017 + 0.008$. Unlike the consecutive strikes parameter $K_u$, the hit count $\Psi_u$ is not reset upon each successful authentication. DALock throttles $u$’s account if the “hit count” exceeds $\Psi$ (i.e., $\Psi_u \geq \Psi$) or if there are too many consecutive mistakes (i.e., $K_u \geq K$).

If an incorrect password guess $pw$ is overly popular this will cause $\Psi_u$ to rapidly reach the threshold so that the account can be locked.

Now the throttling mechanism will kick if either the hit count or the consecutive strike count reaches our thresholds i.e., $\Psi_u \geq \Psi$ or if $K_u \geq K$ and the user will be required to take corrective action(s) to unlock the account. DALock is fully compatible with a wide variety of policies. For example, we could require the user to resend their password, authenticate a request to unlock the account via e-mail/phone and/or solve CAPTCHA challenges. We stress that when a user attempts to login with a password $pw$ the authentication server is able to distinguish between the following cases (1) Account locked/throttled: if $\Psi_u \geq \Psi$ or if $K_u \geq K$, (2) Correct Login: if the guessed password matches the user password i.e., $pw = pw_u$ and the account is not locked\(^2\), or (3) Incorrect Password: the account is not locked but the password is incorrect. We remark that StopGuessing [36] necessarily blurs the distinction between cases (1) and (3), but this can induce a usability cost, e.g., an honest user might be annoyed if they were repeatedly informed that their password is incorrect when, in reality, the account is actually locked.

We use the notation $(K, \Psi)$-DALock to denote DALock instantiated with hit-count threshold $\Psi$ and consecutive strike threshold $K$. Observe that when $\Psi = \infty$, the authentication server is actually running the classical $K$-strikes lockout policy. In most of our experiments we will set $K = 10$ when instantiating DALock and tune $\Psi$ to balance security and usability. The hope is that by tuning $\Psi$ we can achieve (1) stronger security than both the classical $K = 3$-strikes mechanism and $K = 10$-strikes mechanism, and (2) usability superior to the $K = 3$ mechanism and comparable to the $K = 10$ mechanism.

To deploy DALock with a finite hit-count parameter $\Psi$, 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 $p(pw, \text{Estimator})$ to denote the estimated popularity (probability) of a password $pw$ estimated by the estimator $\text{Estimator}$, e.g., given a count sketch $\sigma$ we would use $p(pw, \sigma) = \frac{\text{Estimate}(pw, \sigma)}{\text{TotalFreq}(\sigma)}$.

Remark: One could optionally consider initializing the hit count parameter $\Psi_u$ 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 when $\Psi_u$ reached $\Psi$. Similarly, a user with a strong password might be awarded by setting $\Psi_u = 0$ so that the throttling mechanism will not be activated as quickly. However, because $\Psi_u$ and $K_u$ are stored on the authentication server, this would signal information about the strength of $pw_u$ to an offline attacker. While this seems undesirable, a recent counter-intuitive result showed that noisy strength signals can actually help deter a rational utility maximizing password cracker [1] if the signaling scheme is tuned appropriately. Thus, it is possible that a noisy (randomized) mechanism to tune $\Psi_u$ based on the strength of the user’s password could help deter offline attackers. Alternatively, if one is willing to implement a silent lockout policy where the user cannot distinguish between an incorrect guess and a locked account, it would be possible to encrypt the hit-count $\Psi_u$ using a key derived from the user’s password [13, 36].

4.2 DALock Authentication Server

To implement DALock, we need an efficient way to estimate the probability $p(pw)$ of each incorrect password $pw$. We consider several instantiations of this frequency oracle. One option is to use password strength meters such as ZXCVBN [47] or more sophisticated password cracking models [29, 39]. 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. Hereley and Schechter [35] proposed using the count sketch

\(^2\) 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.
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 $\sigma_{dp} \leftarrow \text{DP}(\epsilon, \sigma)$ by adding Laplace Noise to preserve $\epsilon$-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 [35] and/or to help identify IP addresses associated with malicious online attacks [36]. 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 [2]. Our model for users’ mistakes is informed by recent empirical studies of password typos [12, 13] 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 $p(\cdot)$ is implemented. We consider two concrete implementations: password strength models [29, 39, 47] (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 $p(\cdot)$. We leave the question of tuning DALock to protect against targeted online attackers [45] as an important direction for future research. We elaborate on each of these key model components below. We begin with an overview of the empirical datasets $\mathcal{D}$ that we used in our experiments.

5.1 Experimental Datasets

In this work, we use multiple real-world password datasets. See Table 1 for a summary of each dataset including (1) the total number of unique passwords in the dataset, (2) the total number of user accounts in the dataset, (3) the probability of the most popular password, and (4) the cumulative probability of the top 10 passwords. Except for the differentially private Yahoo! frequency corpus, which was collected [6] and publicly released [3] with permission from Yahoo!, each dataset is the result of a data breach. We remark that the Yahoo! frequency corpus does not contain any plaintext passwords, so we did not use password strength models in our experiments involving this dataset.

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

Table 1. Summary of dataset

Each dataset defines an empirical password distribution. In each of our experiments, we assume that this distribution matches the real (unknown) user 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 password in the distribution will almost certainly closely match the empirical probability.

**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 \( D_U = \{pw_1, \ldots, pw_N\} \) of \( N \) passwords which can be compressed into pairs \( (pw, F(pw, D_U)) \) where \( F(pw, D_U) \) denotes the number of times the password \( pw \) occurs in the dataset \( D_U \). Each dataset \( D_U \) induces an empirical distribution over users’ passwords where the probability of sampling each password \( pw \) is simply \( \frac{F(pw, D_U)}{N} \).

**Simulating Password Choices** Each simulated user \( u \) in our experiment samples 6 different passwords \( pw_u^0, \ldots, pw_u^6 \) from the empirical distribution and registers with the first sampled password \( pw_u^0 \). The remaining five passwords \( pw_u^1, \ldots, pw_u^5 \) intuitively represent the user’s password for other websites and will be used to simulate recall errors (see Section 5.2.2).

**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 [5] to simulate users’ password selections under this restriction. In particular, we use rejection sampling to avoid sampling one of the top \( B \) passwords. Equivalently, we can let \( D_{U,B} \) denote the dataset \( D_U \) with the \( B \) most common passwords removed and sample from the empirical distribution corresponding to the updated dataset \( D_{U,B} \).

#### 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 \( 0 < t^i_1 < t^i_2 < \cdots < 4320 = 180 \times 24 \) where each \( t^i_u \in \mathbb{N} \) represents the time (in hours) of the \( i \)th user visit. Following prior works (e.g., see [2, 25]), 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 \( T_u = E[t_{i+1} - t_i] \). 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.

#### 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 [12, 13] 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 edit distance 2 of the original passwords.

Accordingly, in our simulation we set the mistake rate to be 7.5%, i.e., when simulating each login attempt, the user will enter the correct password with probability 92.5%. Otherwise, we simulate the user’s error(s) — either a recall error or a typo or both. In our simulations of user errors we first flip a biased coin to determine whether to simulate a typo (68%) or a recall error (32%). To simulate a recall error, we randomly select...
one of the user’s five alternate passwords to model a user who forgot which of their passwords was associated with this particular account (the user may additionally mistype this password). When simulating different types of typos (capitalization errors, substitution errors, insertion/deletion errors) we rely on empirical password typo data from [12, 13]. We refer an interested reader to Appendix A for a more detailed discussion of our mistake model, including a flow chart (see Figure 6) and more fine-grained typo statistics. If the login attempt is incorrect the simulated user will repeat the above process until s/he is successful or until the account is locked.

**Remark:** To study the throttling effects of DALock, we do not simulate users who completely forget their passwords (i.e., meaning that the probability of remembering the correct password is non-zero during each login attempt) 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 using a stored password. 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 incrementing both the hit-count $\Psi_u$ and the strike count $K_u$. Alternatively, the authentication server could store an encrypted cache [13] of failed login attempts using public-key cryptography. The encrypted cache could only be decrypted when the user authenticates with the correct password and could be used to avoid unnecessarily incrementing $\Psi_u$ due to repeated mistakes with the same outdated password.

### 5.3 Modeling the Authentication Server

We model an authentication server running $(K, \Psi)$-DALock with various $K$ and $\Psi$ settings. Each time a user $u$ (or attacker pretending to be $u$) failed to login, the authentication server updates the parameters $\Psi_u$ 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 $\Psi$, 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 $p(pw; \text{Estimator})$ to denote the estimated popularity (probability) of a password $pw$ estimated by the estimator $\text{Estimator}$, e.g., given a count sketch $\sigma$ we would use $p(pw, \sigma) = \frac{\text{Estimate}(pw, \sigma)}{\text{TotalFreq}(\sigma)}$.

#### 5.3.1 Differentially Private Count Sketch Estimator

The first instantiation of $p(\cdot, \cdot)$ we consider is to build a count sketch estimator $\sigma_{\text{DP}} = \text{Add}(\mathcal{D}_U, \sigma)$ from the dataset $\mathcal{D}_U$ directly. The authentication server would update the count sketch with the new password each time a new user registers. 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 $\sigma_{\text{DP}}$ 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.UsesStrongpwd!” then we would expect that the true frequency of this password is $F(pw, \mathcal{D}_U) = 1$ (resp. $F(pw, \mathcal{D}_U) = 0$). 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 $\epsilon$-differentially private estimator $\sigma_{\text{DP}} = \text{DP}(\epsilon, \sigma)$ in our experiments. During initialization, we add Laplace noise to the count sketch where the noise parameter scales with $\frac{d+1}{\epsilon}$. In our threat model an attacker can obtain a single snapshot of the differentially private count sketch. In above example, differential privacy ensures that — up to a multiplicative advantage $e^\epsilon$ — 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.UsesStrongpwd!”.

Notice that lower values of $\epsilon$ 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[3], $\epsilon \geq 2$ for collecting users’ information [38], and $\epsilon \geq \ln 81$ for RAPPOR [17, 46].
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 $\sigma_{r\%} = \text{Add}(\mathcal{D}_u, \sigma)$ to denote the count sketch trained on the subsampled data. The question is whether $\sigma_{r\%}$ can be as effective as $\sigma$ 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 $\mathcal{D}_u$ is already small. (e.g., bfield). On the positive side, if one picks an adequate sampling rate (e.g., 10%) or the original dataset size is sufficiently large (e.g., 000webhost), then $\sigma_{r\%}$ can perform nearly as good as $\sigma$.

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 [5]; On the other hand, banning too many of them may raise a usability concern – a large portion of users will not be able to select their preferred password (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 $p(pw, \sigma)$ is already too high, then the user will be asked to pick a different password [35].

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 $\mathcal{D}_u$ 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 [5]. 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., $\sigma_{\mathcal{D}_{\mathcal{U},B}} = \text{Add}(\mathcal{D}_{\mathcal{U},B})$.

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 [47] as prior empirical studies demonstrate that it is one of the most accurate password strength meters [22]. In addition, we used the Password Guessing Service (PGS) [29, 39] 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 [39]. For example, if a password $pw$ had a guessing number $g$, we might estimate that $p(pw_i) = 1/g$. 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 $\sum_{i=1}^{10000} p(pw_i) \geq 1$ 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 $\sum_{i=1}^{\max(10^4,B)} p(pw_i) = 1$ where $B$ is the number of banned passwords. When $B \geq 10^5$ we avoid submitting too many requests to PGS by sampling 20,000 users’ passwords from $\mathcal{D}_{\mathcal{U},B}$ to estimate $\sum_{i=1}^{\max(B)} p(pw_i)$.

Service-Specific Passwords One advantage of a differentially private count sketch is that it can properly account for the popularity of service-specific passwords.
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 com-
plete description of the DA Lock mechanism (e.g., K and
Ψ). In particular, for any password pw, we assume that
the attacker knows both the true probability P(pw) and
the estimated probability p(pw). We also assume that
the attacker is given the complete sequence of login
times t1 ≤ t2 ≤ . . . ≤ 24 × 180 for each user u over a
180-day time span as well as the outcome of each, e.g.,
at time t 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 se-
cquence of login times and outcomes. We use Ku,t (resp.
Ψu,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, a large online attacker will be able to approxi-
mate P(pw) and p(pw) over time by interacting with the
DA Lock 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 out-
comes 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 acci-
dentally locking the user’s account based on the number
of consecutive incorrect login attempts.

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 p(pw)/P(pw) is small so that the increased hit count is
smaller than intended when it fails. We formalize the at-
tacker’s optimal strategy in terms of the Password Knapsack
problem (PK). Unsurprisingly, the password knapsack
problem turns out to be NP-hard (see full version of
the paper), but there are several heuristic algorithms
the A can use to achieve nearly optimal results in prac-
tice.

Supposing that the attacker wishes to avoid lock-
down the user’s account before a particular time
then the cumulative (estimated) probability of all
guesses submitted before that time should be at most
Ψu,t := Ψ − Ψu,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 Ku,t ≥ K at some time t ≤ t. (Recall Ku
resets whenever u login successfully).

Fixing a time parameter t, the attacker’s goal is to
find a subset Si ⊆ P of M(t) passwords to guess such that

$$\sum_{pw \in S_i} P(pw) \leq \Psi'_{u,t}.$$  (1)

After checking the passwords in Si the attacker can still
guess one more password pwhold ∉ Si before the account
is locked down. Given a set Si and a holdout password
pwhold ∉ Si the probability that the attacker succeeds is

$$P(pw_{hold}) + \sum_{pw \in S_i} P(pw).$$  (2)

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

Password Knapsack Problem Given a pass-
dword dictionary {pw1, . . . , pwn} we formally define the
Password Knapsack (PK) problem as the following in-
teger program with indicator variables si ∈ {0, 1} and
li = {0, 1} for each password pw. 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' (Ψ'_{u,t})

$$\max \sum_i (s_i + l_i) \cdot P(pw_i)$$

s.t. (1) $\sum_i s_i \cdot p(pw_i, \sigma) \leq \Psi'$
(2) $\sum_i s_i \leq M$
(3) $\sum_i l_i \leq 1$
(4) $\forall i l_i + s_i \leq 1$
(5) $\forall i, s_i, l_i \in \{0, 1\}$

Intuitively, setting si = 1 means pw_i is selected to be
placed in the “password knapsack” S ⊆ P, i.e., to be
used for dictionary attack. Setting li = 1 indicates that
password pw_i is used as a holdout password. The con-
straints ensure that |S| ≤ 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 $\Psi_{u,t} := \Psi - \Psi_{u,t}$ for each time $t \leq 24 \times 180$ and solve the corresponding Password Knapsack problem. Given optimal solutions $(pw_{\text{hold},t}, S_{t})$ 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 full version of the paper 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) [27] 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: Dantzig’s Algorithm Based [14] approach (DAB) and Feasible Most Promising Password First approach (FMPPF).

DAB sorts passwords $P_{\Pi} = \{pw_{2}, \ldots, pw_{n}\}$ based on the how much they are underestimated, i.e., $p(pw_{i}) < p(pw_{j})$, 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 $\Psi'$. FMPPF sorts the passwords differently by using the true probability $P(pw_{i})$ and FMPPF simply selects password $pw_{i}$ in sorted order. More detailed discussion can be found in the full version of our work. Intuitively, FMPPF (resp. DAB) will perform better when $M$ (resp. $\Psi'$) 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 $p(pw_{i}) \approx P(pw_{i})$, 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 $P(pw_{i})$ 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 simulated 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 $A$ (see section 5.4) to the simulation and measured the fraction of user passwords $A$ 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 $\Psi$ 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 $\Psi$ 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, which offers great security and the 10-strikes mechanism, which offers close-to-zero unwanted lockout, (recommend by Brostoff et al. [8] to improve usability) as baselines for comparisons. We exclude the K=5-Strike mechanism [33] from our results since K=3 offers strictly better security and K=10 offers strictly better usability than 5-Strikes. Our simulations demonstrate (10, $\Psi'$)-DALock achieves better security than 3-strikes and comparable usability as 10-strikes. Notice that 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 $\Psi$ 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 $\Psi$ to measure the % of cracked passwords (resp. % locked users) when the simulation includes (resp. excludes) an online attacker. Legend entries are in the for-
Fig. 3. Usability/Security Tradeoff of DALock with \((B = 10^4)\)

Fig. 4. Security Measurement of DALock

Fig. 5. Usability Measurement of DALock

mat 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 \(\Psi\). Figure 3 demonstrates that DALock offers a superior usability/security tradeoff when instantiated with a suitable frequency 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 \((0.1\text{-CS-all})\). We use the notation \(\epsilon\text{-CS-all}(\text{resp. } \epsilon\text{-CS-X\%})\) to refer to an \(\epsilon\)-differentially private count sketch trained on the entire dataset \(D_U\) (resp. a dataset \(D_{U\%}\) obtained by sampling X\% of user passwords from \(D_U\)). Training the differentially 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 $\Psi$ to improve security, but this occasionally results in inferior usability.

6.2 Impact of Ban-List Size

We demonstrate the usability/security impact of the ban-list size $B \in \{0, 5, 10, 100, 1000, 10000, 100000\}$ holding the other DA\text{}lock parameters $k = 10$ and $\Psi$ 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 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 DA\text{}lock: 3-strikes($k:3$, $\Psi$: $\infty$), 10-strikes($k:10$, $\Psi$: $\infty$), 0.1-CS-all($k:10$, $\Psi:2^{-10.0}$), 0.1-CS-5%($k:10$, $\Psi:2^{-10.0}$), ZXCVBN($k:10$, $\Psi:2^{-9.0}$), Min($k:10$, $\Psi:2^{-7.0}$), Hashcat($k:10$, $\Psi:2^{-9.0}$), Markov($k:10$, $\Psi:2^{-8.0}$), NeuralNet($k:10$, $\Psi:2^{-8.0}$), and PC\text{}FG($k:10$, $\Psi:2^{-8.0}$). Legend entries are in the format FrequencyOracle($k,\Psi)$ where $k$ and $\Psi$ are the DA\text{}lock 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 DA\text{}lock for handpicked $\Psi$ parameters (e.g., $\Psi = 2^{-10}$ for differentially private count sketch). Additional plots in the appendix explore the impact of the privacy budget $\epsilon$ 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 b\text{}field datasets while results for the brazzers, csdn and clic\text{}s\text{}sense datasets can be found in the appendix (see Figures 8 and 9).

Usability Firstly, Figure 5 clearly demonstrates the unwanted lockout rate of $(10,\Psi)$-DA\text{}lock 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 DA\text{}lock. e.g., from 2.56% to 0.08% for 0.1-CS-all after banning 1000 passwords from b\text{}field. 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 DA\text{}lock implementations achieve 10-strikes level lockout rate, i.e., $\approx 0\%$. While the unwanted lockout rate for DA\text{}lock 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\text{}-based DA\text{}lock 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 b\text{}field, 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 $\Psi$. However, for the same reason, subsampling can negatively impact security when the dataset was already small (see section 6.1).

Security When we implement DA\text{}lock with a differentially private count sketch ($\epsilon=0.1\text{-CS-all}(k,\Psi)$ 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 $(10,\Psi)$-DA\text{}lock will always outperform the traditional 10-strikes mechanism, which is equivalent to $(10, \infty)$-DA\text{}lock. 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 DA\text{}lock (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 b\text{}field, then the attacker compromises 0.536% (resp. 0.08%) of user accounts when adopting the traditional 3-strikes mechanism (resp. DA\text{}lock with a differentially private count sketch). Recall that the usability of DA\text{}lock 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 re-
sult 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.

An observant reader might notice that in Fig 4 (bfield) the % of compromised accounts increased in some plots when the ban list size increased from B=10^4 to B = 10^5. The explanation for this anomalous result is twofold. First, the bfield dataset is small enough (0.5 million accounts) that removing the top 100K passwords substantially increases the probability of the remaining passwords in the empirical distribution. Second, as discussed at the bottom of Section 5.3.2 normalization for model based frequency estimators also shifts at B = 10^5.

6.3 Summary and Discussion

We find that CS/ZXCVPN-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 ZXCVPN 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 and deploying \((10, Ψ)\)-DALock with \(K = 10\) strikes and hit count parameter \(Ψ \in [2^{-8}, 2^{-10}]\). For smaller organizations, we recommend implementing DALock with ZXCVPN e.g., ZXCVPN(\(K : 10, Ψ : 2^{-9}\)).

Limitations Our empirical usability and 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. Another future direction of study is to conduct a longitudinal user studies to confirm the ecological validity of the simulated usability results. Finally, we only maintain the synchronized state \((K_u, Ψ_u)\) on each authentication server. 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.

Locking Accounts vs Blocking IPs In our simulated evaluation of DALock we assume that each user account \(u\) is locked if the hit count \(Ψ_u\) exceeds the threshold \(Ψ\) (or if the consecutive strike threshold \(K\) is reached). An alternative (more lenient) implementation of DALock would instead maintain the state \((K_{u,ip}, Ψ_{u,ip})\) for each distinct user/IP pair \((u, ip)\) where \(Ψ_{u,ip}\) tracks the total hit count (resp. consecutive incorrect guesses) for all guesses submitted from the IP address \(ip\) against user \(u\). Under this alternate approach we could block a (malicious) IP address from attempting to login to account \(u\) once \(Ψ_{u,ip}\) exceeds the threshold \(Ψ\). One advantage of this approach is that it is less vulnerable to denial of service attacks and we are less likely to lockout the legitimate user who will (most likely) have a different IP address. Furthermore, this approach may be easier to implement in a distributed setting as the servers do not need to synchronize the state \((K_u, Ψ_u)\) for each user — instead each authentication server would independently maintain the value \((K_{u,ip}, Ψ_{u,ip})\) for IP addresses in its service area. On the downside blocking individual IP addresses instead of accounts allows an distributed online attacker to launch coordinated attacks from multiple different IP addresses (e.g., through botnets) increasing the the risk to each user account.
Acknowledgements

Jeremiah Blocki was supported in part by NSF CAREER Award #2047272 and by NSF Award CNS#1755708. Wuwei Zhang was supported in part by NSF Award #2046707. The authors thank anonymous PC members and shepherd Daniel Votipka for feedback that improved the presentation of this paper.

References


A Simulating Users’ Mistakes

In this section, we include the complete flowchart used to simulate typos and recall errors — see figure 6. The first node in the flowchart simulates recall errors, and we set this probability to be 2.4%, a number we derived based on empirical data from [12, 13]. When simulating a recall error we randomly select $pw'$ from one of the five other passwords we previously selected for our simulated user (the passwords represent the user’s other accounts). At this point in the flow chart we simulate whether or not the user miss-types his intended password (5%) or enters it in correctly (95%) — the number 5% was derived from empirical data collected in [12, 13]. When simulating a typo we further follow the empirical data in Table 2. Notice that our simulated user can make both mistakes, e.g., recall the wrong password $pw'$ and misstype the password $pw'$. For example, suppose that the user’s actual password is letmein. The simulated user will recall the correct password and type it correctly with probability $0.976 \times 0.95 \approx 0.927$, and the simulated user will enter LETMEIN (CAPSLOCK error) with probability $0.976 \times 0.05 \times 0.04 \approx 0.002$. Suppose that the simulated user has 5 other passwords and one of them is 123456. In this case the simulated user would enter 123456 with probability $0.024 \times (1/5) \times 0.95 \approx 0.0046$ — the $(1/5)$ term is the conditional probability of recalling 123456 when simulating a recall error.

<table>
<thead>
<tr>
<th>Typo</th>
<th>Mistake % (Rounded)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapLock On</td>
<td>14</td>
</tr>
<tr>
<td>Shift First Char</td>
<td>4</td>
</tr>
<tr>
<td>One Extra Insertion</td>
<td>12</td>
</tr>
<tr>
<td>One Extra Deletion</td>
<td>12</td>
</tr>
<tr>
<td>One Char Replacement</td>
<td>31</td>
</tr>
<tr>
<td>Transposition</td>
<td>4</td>
</tr>
<tr>
<td>Two Deletions</td>
<td>3</td>
</tr>
<tr>
<td>Two Insertions</td>
<td>3</td>
</tr>
<tr>
<td>Two Replacements</td>
<td>10</td>
</tr>
<tr>
<td>Others</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 2. Typo Distributions[13]

B More Experimental Results

In this section, we provide more detailed experimental results for readers to understand the underlying details of DALock. Figure 7 plots the unwanted locked rate (usability) vs. compromised accounts (security) for various instantiations of DALock as the hit-count parameter varies: $\Psi \in \{2^{-8}, 2^{-9}, 2^{-10}, 2^{-11}, 2^{-12}\}$. Note: The plot is similar to Figure 3 except that the listsize is $B = 10^3$ in Figure 7 while $B = 10^4$ in Figure 3. Additional experimental plots are available in the full version of the paper including usability/security plots for $B \in \{0, 10, 100, 10^4, 10^5, 10^6\}$. In the full version, we also evaluate the performance of the differentially private count sketch as the privacy budget $\epsilon \in \{0.1, 0.5, 1.0, \infty\}$ and subsampling rate $\{1\%, 5\%, 10\%, 100\%(all)\}$ vary. Intuitively, $\epsilon = 0.1$ provides the strongest privacy guarantee and $\epsilon = \infty$ indicates no differential privacy. Briefly, we found that the differentially private count-sketch performs sufficiently well even when using the smallest privacy budget $\epsilon = 0.1$. Thsu, due to space limitations,
we only show our results with the subsampling rates 5% and 100% (all) fixing \( \epsilon = 0.1 \) (strongest privacy).

Figure 8 (resp. Figure 9) plots the number of compromised accounts (resp. number of unwanted lockouts) vs. banlist size to illustrate the security (resp. usability) of DALock under various instantiations. The plot is similar to Figures 4 and 5 from the main body except that we include additional password datasets and we evaluate DALock with different hit-count parameters.

We begin by discussing the pros and cons of each frequency oracle based on our results, and then provide our recommendations on how to deploy DALock.

### PCFG/NeuralNet/Markov/HashCat/Min

When adopting one of these models as a frequency oracle for DALock one can achieve better usability than the 3-strikes mechanism as long as \( \Psi \geq 2^{-9} \). When the ban-list size \(|B|\) is small we find that the the security is also improved. However, as the ban-list size increases compromised account rate for 3-strikes drops slightly below DALock when instantiated with a frequency oracle based on guessing numbers derived from these models (PCFG/NeuralNet/Markov/HashCat/Min) — security is still better than the classical 10-strikes mechanism. Based on these observations we recommend against instantiating DALock with these frequency oracles as one can obtain better performance with other frequency oracles.

### ZXCVBN

If it is not feasible to implement DALock with a differentially private count sketch we recommend deploying DALock with \( \Psi = 2^{-9} \) using ZXCVBN as our frequency oracle. We find that ZXCVBN offers better security and usability in comparison to the classical 3-strikes mechanism even when the banlist size \( B \) is larger. 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, we recommend setting \( \Psi \geq 2^{-9} \) to avoid uncessary lockouts (usability). When setting \( \Psi \geq 2^{-9} \) and using a moderately size banlist (e.g., \( B = 10^5 \)) we find that usability is close to the much-less-secure 10-strikes mechanism while security is better than the much-less-usable 3-strikes mechanism.

### Differentially Private Count Sketch

We find that usability/security performance of DALock is best when we implement with a count-sketch frequency oracle provided that we have sufficient data to train the differentially private count-sketch. Tuning \( \Psi \) 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 - \( D_{U|5\%}, D_{U|10\%}, \) and \( D_{U|10\%} \) - in addition to \( D_U \). Our simulation results show that lower sampling rates can adversely impact security as \( A \) can take advantage of underestimated passwords. We also observe that 0.1-CS-10%/0.1-CS-5%/0.1-CS-1% are nearly as accurate as 0.1-CS-all when we have more than 2/6/32 millions users in the \( D_U \)(see clixsense/csdn/RockYou). This result empirically shows organizations need approx. 0.2-0.3 million users to train a sufficiently accurate differentially private Count Sketch. In summary, if the organization has more than 0.3 million users we recommend deploying DALock with a \( \epsilon = 0.1 \)-differentially private count sketch and \( \Psi \in [2^{-8}, 2^{-10}] \).

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (K, \Psi) ) - DALK</td>
<td>DALK with strike threshold ( K ) and hit count threshold ( \Psi )</td>
</tr>
<tr>
<td>( A )</td>
<td>Adversary</td>
</tr>
<tr>
<td>( U )</td>
<td>A set of users</td>
</tr>
<tr>
<td>( u )</td>
<td>A user ( u \in U )</td>
</tr>
<tr>
<td>( P )</td>
<td>The set of all potential user passwords</td>
</tr>
<tr>
<td>( \mathcal{D}_U \subseteq P )</td>
<td>a multiset of ( N ) sampled passwords for users ( u_1, \ldots, u_N \in U )</td>
</tr>
<tr>
<td>( pw_u )</td>
<td>User ( u )'s password</td>
</tr>
<tr>
<td>( pw_u )</td>
<td>The ( r )’th most likely password in ( \mathcal{D}_U \subseteq P )</td>
</tr>
<tr>
<td>( CS )</td>
<td>Count (Median) Sketch data structure</td>
</tr>
<tr>
<td>( F(pw, \mathcal{D}_U) )</td>
<td>Frequency of password ( pw ) in dataset ( \mathcal{D}_U )</td>
</tr>
<tr>
<td>( P(pw) )</td>
<td>Estimated frequency of password ( pw )</td>
</tr>
<tr>
<td>( \text{Estimate}(pw) )</td>
<td>Estimated probability of password ( pw )</td>
</tr>
<tr>
<td>( \text{p}(pw) )</td>
<td>Estimated probability of password ( pw )</td>
</tr>
<tr>
<td>( \Psi )</td>
<td>Hit count threshold</td>
</tr>
<tr>
<td>( \Psi_u )</td>
<td>Cumulative hit count threshold on ( u )'s account. The account gets locked out if ( \Psi_u ) exceeds ( \Psi )</td>
</tr>
<tr>
<td>( K )</td>
<td>Traditional strike threshold.</td>
</tr>
<tr>
<td>( K_u )</td>
<td>Cumulative strike threshold on ( u )'s account. The account gets locked if ( K_u ) exceeds ( K )</td>
</tr>
</tbody>
</table>

| Table 3. Notation Summary |

### C Encrypted Password Cache

Previously we mentioned that it may be desirable to maintain an encrypted cache of incorrect login attempts so that we can avoid unnecessarily incrementing \( \Psi_u \) if
Fig. 7. Usability/Security Trade-off (Banlist Size = 1000)

Fig. 8. Security Measurement of DALock (All Datasets)

Fig. 9. Usability Measurement of DALock (All Datasets)
there are many login attempts with the same incorrect password e.g., the user’s old password. We want to ensure that the encrypted password cache can only be decrypted when the user provides the correct password. In this section we briefly review a solution of Chatterjee et al.[13] as part of their personalized typo corrector.

As described in [13], we can generate a public/private key pair \((pk_u, sk_u)\) for each user \(u\) and derive a symmetric secret key \(K_u = \text{KDF}(pw_u)\) from the user’s password \(pw_u\). The authentication server will store the public key \(pk_u\) along with a symmetric encryption \(c = \text{Enc}_{K_u}(sk_u)\) of the corresponding secret key \(sk_u\). Now whenever we see an incorrect login attempt \(pw'\) we can encrypt \(pw'\) using the public key \(pk_u\) and the resulting ciphertext \(c' = \text{PKEnc}_{pk_u}(pw')\) can be decrypted when the user provides the correct password i.e., given \(pw_u\) we can recover \(K_u = \text{KDF}(pw_u)\), decrypt \(sk_u = \text{Dec}_{K_u}(c)\) to recover the secret key \(sk_u\) and finally decrypt \(pw' = \text{PKDec}_{sk_u}(c')\). However, since \(sk_u\), \(K_u\) and \(pw_u\) are not stored on the server the encrypted cache could only be decrypted when the user authenticates with the correct password. The encrypted cache could be used as part of a personalized typo corrector [13] and could also be used to avoid penalizing repeat mistakes [13, 36]. 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.