Template for ACM CCS

Anonymous Author(s)

ABSTRACT

Large-scale online password guessing attacks are wide-spread 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 within a fixed period of time (e.g., 24 hours). Selecting the value of K induces a classic security-usability tradeoff. 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 than the password guesses of an online attacker, we introduce the notion of a *password distribution aware* lockout mechanism to reduce user annoyance while minimizing user risk. As the name suggests, our system is designed to be aware of the frequency and popularity of the password used for login attacks while standard throttling mechanisms (e.g., K-strikes) are oblivious to the password distribution. In particular, we maintain an “hit count" for each user which is based on (estimates of) the cumulative probability of *all* login attempts for that particular account. A user will only be locked out when this hit count is too high. To minimize user risk we use a differentially private CountSketch to estimate the frequency of each password and to update the “hit count" after an incorrect login attempt. To empirically evaluate our new lockout policy we generate a synthetic dataset to model honest user logins in the presence of an online attacker. The result of our analysis on this synthetic dataset strongly support our hypothesis that distribution aware lockout mechanisms can simultaneously reduce both user annoyance *and* risk.

CCS CONCEPTS

• **Security and privacy** → Use https://dl.acm.org/ccs.cfm to generate actual concepts section for your paper;

KEYWORDS

template; formatting; pickling

1 INTRODUCTION

1.1 Background

Throttling parameter is a double-edged sword to dealing with for classic authentication system such as K-strike. As its name suggests, K-strikes mechanism lock out an user's account if K consecutive incorrect passwords are attempted. For the purpose of accounts' security, system administors are advised to use stricter values such as 3 and 5 to defend aginst online dictionary attacks. On the other hand, more flexible values are demended to grant users more chances to entering their passwords. Ultimately, there is always a trade-off between security and usability (or fault-tolerant) for configuring K, since all login attempts are treated equally.

One fundamental motivation for introducing throttling mechanism is to prevent password guessing attacks. It is not a mystery that within a few attempts of *popular passwords*, adversaries are able to compromise a significant amount of accounts.[35][43][41]. Adversaries are encouraged by the fact that users tend to choose (common) weak passwords. Dictionary attackers typically hold concret password dictionaries, for example - passwords are sorted based on frequencies which allows them to arrange optimal attacks based on the chance of success, i.e. popularities of the passwords.

The usability is a raising demand in recent decade for K-strike throttling. A survey[17] in 2018 shows that each email address is associated with 130 accounts on average. Nowadays, It becomes natural for a user to enter incorrect passwords (perhaps multiple times) before recalling the correct one. In addition, due to other factors such as keyboard failures, there is a non-negligible probability for users to misenter their (correctly recalled) passwords. Unfortunately, to reduce risk of being compromised by the afore-mentioned dictionary attacks, service providers are conservative and meticulous about the choice of K.

Existing solutions for improving users' experience without comprmisation of security exist; however, to the best of our knowledge, none of them serves as a replacement for the K-Strike throttlings. Concisely speaking, there are two major approaches to enhance usability - 1) Increasing the difficultiness of authentication process, and 2) Identify Honest mistakes (and tolerate those). Captcha[36], Two (or multi) factor authentication[1], Biometrics identifications are classic examples of the former apporach. Such methodology somewhat mitigate the issue of nwanted locked outs, yet they are not perfect replacement (or “add-ons") of a rigious authentication throttling system. Recent works[14][13] on second approaches have achieved satisfactory results on tolerating typos. Even so, there is plenty room for improvement due to their coverage (for the types of mistakes).

There is a contemporary “self-contradictory" demand for the next generation of throttling mechanism as internet services are becoming inseparaetable companions of us. To defend against dictionary attacks, one wants to introduce a stricter policy for attackers. At the same time, users' should not be overwhelmed by the headaches comes from rate limiting such as password resetings.

1.2 Contributions

To achieve better usability without compromising security, we propose a novel password Distribution Aware throttling mechanism: DALock. On the usability side, DALock tolerates users' honest mistakes by granting them more chances to enter their correct passwords as demonstrated in **Figure** 1. Surprisingly on the security side, dictionary attacker does not benefit from DALock and still subject to limited chance of guessing (**Figure** 2). In this work, we theoretically proved that adversaries are challegned by a computational difficult task to perform optimal attacks. We further empirically measure the performance of DALock for both security and usability on three real datasets.

In this work, we focus on the online dictionary attack scene. We assume that dictionary attackers are trying to optimize their

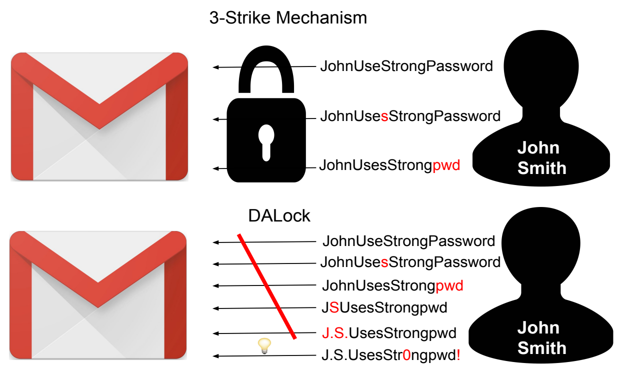

Figure 1: Usability Comparaison

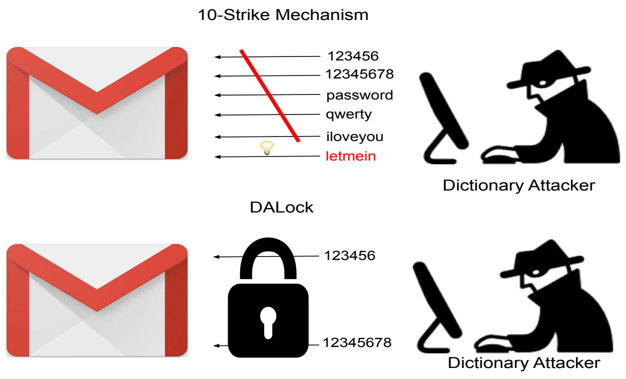

Figure 2: Security Comparaison

chance of success globally.. i.e., the adversaries are trying to break as many accounts as possible. In this work, we assume they have reasonable efficient computational powers, for example, a bitcoin mining machine, to perform non-trivial task easily (problems not in NP). On top of that, the adversaries have precise knowledge of the distribution of users' passwords and deploed security parameters of DALock mechanism. More detailed description of adversarial model can be found in **section** 8.

2 RELATED WORKS AND BACKBROUNDS

2.1 Authentication Throttling

K-strike Mechanism K-strike mechanism is a straight-forward implementation for authentication throttling. As its name suggests, throttling occurs whenever K consecutive incorrect login attempts are detected. To reduce the cost of expensive overhead caused by unwanted throttling, Brostoff [9] et.al suggests setting threshold K to be 10 instead of 3. They argue the increment risk is limited when strong password policy is enforced. However, this argument is challenged by empirical analyses of password composition policies [28][7]. 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. [28]

Feature-Based Mechanism To improve performance, modern throttling mechanisms[38][25] often times use features such geographical location, IP-address, device information, and etc in addition to the correctness of attempting password. These features can be used to train sophisticated machine learning models to help distinguish between malicious and benign login attempts [21]. One can combine those models with a rigious throttling system for a better performance.

Password-Distribution Aware Throttling In an independent line of work Tian et al. [51] developed an IP-based throttling mechanism which exploits differences between the distribution of honest login attempts and attacker guesses. In particular, they propose to "silently block" login attempts from a particular IP address a if the system detects too many popular passwords being submitted from that IP address. In more detail StopGuessing uses a data-structure called the binomial ladder filter [40] to (approximately) track the frequency $F(p)$ of each incorrect password guess. For each IP address the StopGuessing protocol maintains an associated counter $I_a = \sum_{p \in \mathcal{P}} F(p)$ where $\mathcal{P}$ is a list of incorrect password guesses that have been (recently) submitted from that IP address — I_a can be updated without storing $\mathcal{P}$ explicitly. Intuitively (and oversimplifying a bit) if $I_a > T$ then login attempts from address a 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 $T(F(u_p))$ based on the strength $F(u_p)$ of the password u_p of user u. Now if $I_a > T(F(u_p))$ then the system will silently reject any password from address a. Both StopGuessing and DALock exploit differences between the distribution of user passwords and attacker guesses. One of the key difference is that StopGuessing focuses on identifying malicious IP addresses (by maintaining a score I_a) while DALock focuses on protecting individual accounts by maintaining a "hit-count" para I_u for each user. There are several other key differences between the two approaches. 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 a 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. See Section **section** 9 for additional comparisons.

2.2 Passwords

Password Distribution Password distribution naturally represents the chance of success in the setting of statcial guessing attacks. Password distribution has been extensively studied since last decades[20][31]. Using leaked password corpora[37][30][8] is a straight forward way to describe the distribution of passwords. In recent works of Wang et al. [47][46][48] argue that password distributions follows Zipf's law i.e., leaked password corpora nicely fit Zipf's law distributions. Blocki et al. [6] later found that Zipf's law nicely fits the Yahoo! password frequency corpus [5, 8].

Password Typos To test the usability of DALock, it's crucial to reasonably simulate users' mistakes. Recent studies[14][13] from Chatterjee et. al have summarized probabilities of making (various of) typos when one enters his or her password based on users' studies. Based on the empirically measured data, they purposed two typo-tolerant authentication without sacrificing security. If fact, such mechanism has already been deployed in industry[52][23].

2.3 Eliminating Dictionary Attacks

Increasing Cost of Authentcation Pinkas and Sanders [36] proposed the use of puzzles (e.g., proofs of work or CAPTCHAs) as a way to throttle online password crackers. CAPTCHAs are hard AI challenges meant to distinguish people from bots [44]. For example, reCAPTCHA [45] has been widely deployed in online web services such as Google, Facebook, Twitter, CNN, and etc. Assuming that CAPTCHAs are only solvable by people, one can mitigate automated online dictionary attacks without freezing users' accounts [10, 11]. However, an attacker can always pay humans to solve these CAPTCHA challenges (cite stats here between \$0.6 and \$2.4 per 1000 CAPTCHA solves). Increasingly sophisticated CAPTCHA solvers [22, 50] powered by neural networks make it increasingly difficult to design CAPTCHA puzzles that are also easy for a human to solve. Golla et al. [24] 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 loosing 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. Schechter et. al [39] show that it is possible to forbid the existence of over popular password by maintaining the password distribution securely. Industrial solutions such as "Have I been pwned?" [26] and "Password CheckUp" [42] " prevent users to choose weak passwords based on data breaches. Password strength meters such as zxcvbn [49] are also been widely deployed to help users choosing stronger passwords.

2.4 Privacy Perserving Aggregate Statistics Releasing.

DALock relies on the distribution of passwords to perform throttling. Storing/Releasing aggregate statistics naively often causes privacy leakage [33] [34]. To answer the challenge, Cynthia Dwork purposed Differential Privacy [18] for aggregated data releasing. Informally speaking, differential private algorithm makes powerful adversaries unable of telling the existence of a record in the dataset. We defer the formal definition of differential privacy to section 3.2. Blocki et.al released the statics of Yahoo! dataset consist of 70 millions of passwords. [5]. Recent work by Naor et at [32] also demonstrates that releasing the distribution of password privately is feasible. In industry, Differential Privacy has been considered as the golden tool for various of tasks [2] [3] [19].

3 PRELIMINARIES

3.1 Count Sketch

Count (Median) Sketch [12] and are its variants are widely used to estimate frequent items in unbounded or extremely large domain. For example, users' passwords [32], user's homepage setting [19], and user's frequently sent emoji [2]. For the purpose of illustration, we "abusively" refer "Count Sketch" to Count Median Sketch purposed by Charikar et al [12]. in 2012. In this section, we formally introduce Count Sketch and it's variants: Count Mean Sketch and Count Min Sketch.

Definition 3.1 (Count (Median) Sketch [15] [12]). A Count Sketch with parameters (ϵ, δ) is represented by a two-dimensional array counts with width w and depth d: count[1, 1] $\cdots$ count[d, w]. Each entry of the array is initially zero. Additionally, d + 1 hash functions

$$h_1 \cdots h_d : \{\sigma^*\} \rightarrow \{1 \cdots w\}$$
$$h_\pm : \{\sigma^*\} \rightarrow \{1, -1\}$$

are chosen uniformly at random from a pairwise-independent family. Finally an integer T is maintained to record the total frequency.

A typical Count Sketch involves the following operations:

Add(p): Given input password word $p = \sigma^*$, CS updates the table as follows:

$$\forall i \in d, CS[d, h_d(p)] \leftarrow CS[d, h_d(p)] + h_\pm(p)$$
$$T \leftarrow T + 1$$

Estimate(p) Given input password word $p = \sigma^*$, CS return the estimate frequency count of p as follows:

$$\text{median}_{\forall i \in d}\{CS[d, h_d(p) \cdot h_\pm(p)]\}$$

Count Mean Sketch differs from Count Sketch in estimation mechanism. As it's name suggests, the frequency is estimated by averaging the value over d (hashed) values.

Estimate(p) Given input password word $p = \sigma^*$, CS_{mean} return the estimate frequency count of p as follows:

$$\text{mean}_{\forall i \in d}\{CS_{mean}[d, h_d(p) \cdot h_\pm(p)]\}$$

Count Min Sketch differ from Count Sketch with the followings

- No $h_\pm$ in Count Min Sketch
- **Add(p):** Given input password word $p = \sigma^*$, CS_{min} updates the table as follows:
$$\forall i \in d, CS_{min}[d, h_d(p)] \leftarrow CS_{min}[d, h_d(p)] + 1$$
$$T \leftarrow T + 1$$
- **Estimate(p)** Given input password word $p = \sigma^*$, CS_{min} return the estimate frequency count of p as follows:
$$\min_{\forall i \in d} \{CS_{min}[d, h_d(p)]\}$$

3.2 Differential Privacy

Differential Privacy [18] is one of the industrial golden standard tools for private aggregated statical releasing as mentioned in 2.4. In this section, we formally introduce the notion of differential privacy and Laplace mechanism, one popular way to implement DP.

Definition 3.2 (ϵ-Differential Privacy [18]). A randomized mechanism $\mathcal{A}$ gives ϵ-differential privacy if for any pair of neighboring datasets D and D', and any $S \in Range(\mathcal{A})$,

$$\Pr[\mathcal{A}(D) = S] \le e^{\epsilon} \cdot \Pr\left[\mathcal{A}(D') = S\right].$$

In this paper we consider two datasets D and D' to be neighbors i.f.f. either $D = D' + p$ or $D' = D + p$, where $D + p$ denotes the dataset resulted from adding the tuple p(a new password) to the dataset D. We use $D \simeq D'$ to denote this. This protects the privacy of any single tuple, because adding or removing any single tuple results in e^{ϵ}-multiplicative-bounded changes in the probability distribution of the output. If any adversary can make certain inference about a tuple 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 computes a function g on the dataset D in a differentially privately way, by adding to $g(D)$ a random noise. The magnitude of the noise depends on GS_g, the *global sensitivity* or the L_1 sensitivity of f. When f outputs a single element, such a mechanism $\mathcal{M}_g$ is given below:

$$\mathcal{M}_f(D) = f(D) + \mathsf{Lap}(\frac{\mathsf{GS}_g}{\epsilon})$$

where

$$\mathsf{GS}_g = \max_{(D,D'):D \simeq D'} ||g(D) - g(D')||_1,$$

and

$$\Pr[\mathsf{Lap}(\beta) = x] = \frac{1}{2\beta} e^{-|x|/\beta}$$

In the above, $\mathsf{Lap}(\beta)$ denotes a random variable sampled from the Laplace distribution with scale parameter β. When f outputs a vector, $\mathcal{M}_f$ adds a fresh sample of $\mathsf{Lap}(\frac{\mathsf{GS}_g}{\epsilon})$ to each element of the vector. This is generally referred to as the *Laplacian mechanism* for satisfying differential privacy.

Differential Private Count Sketch. In our work, we treat each tuple in the database $\mathcal{P}$ is a single password string $p = \sigma^*$. Given a CS of size $d \cdot w$, adding or removing any password from $\mathcal{P}$ can result in at most d + 1 for l_1 norm. Because each p only contributes to d different hashed index and total count T. Therefore the global sensitivity GS is d + 1. To make a single CS release with privacy budget ϵ, one needs to add $\mathsf{Lap}(\frac{d+1}{\epsilon})$ to every index in CS including T.

Existing Privacy-Preserving Password Corpus Relasing Mechanism Naor et.al[32] purposed an algorithm to release password distribution using local differential privacy. In our work, we focused on a centralized version of differential privacy which is expected to have less noisy compare to using local differential privacy. StopGuessing[51] 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 require a more fine grained estimate of a passwords popularity.

3.3 Notation Summary

In this section, we summarize all notations used in this paper across all sections in **Table** 1

Notation	Description
$\mathcal{A}$	$\underline{\mathcal{A}}$dversary
$\mathcal{U}$	The set of $\mathcal{U}$ser
u	A user u $\in \mathcal{U}$
$\mathcal{P}$	The set of all $\underline{\mathcal{P}}$asswords in database
p_u	User u's password p
p_r	Rank r password in $\mathcal{P}$
CS	Count (Median) Sketch data structure
CS(p)	Estimated popularity of p from Count Sketch
w	width (or number of columns) of CS
d	depth (or number of rows) of CS
CS.T	Total frequency counts of CS
h_d	hash function for $\underline{d}^{th}$ row
$\mathsf{h}_\pm$	hash function to compute the sign of a key.
F(p)	Ground Truth Frequency of password p
f(p)	Actual popularity of password p
DP	Differential Privacy
LAP($\frac{d}{\epsilon}$)	Laplace noise with privacy budget ϵ and sensitivity d
Ψ	hit count threshold
K	traditional K-strike threshold

Table 1: Notation Summary

4 THE DALOCK MECHANISM

In this section, we present the DALock mechanism, discuss how DALock might be implemented and the strategies that an attacker might use when DALock is deployed.

4.1 DALock

We first introduce the classic K-strike throttling mechanism. As its name suggests, throttling happens whenever one makes K consecutive mistakes. To achieve this goal, K-strike mechanism maintains a counter K_u for each user u to record how many incorrect attempts were made. Typically, K_u is reset to 0 whenever u logins successfully. Some systems also reset K_u to 0 after a certain amount of time and/or maintain separate counters for different IP addresses. In our experiments we consider the K-strike mechanism where K_u is reset to 0 only after successful login attempts. The mechanism is described formally in the appendix (see **Algorithm** 4).

The DALock mechanism maintains an extra "hit count" variable Ψ_u for each user. Intuitively, Ψ_u measures the total probability mass of all incorrect password guesses submitted for user u. For example, suppose that the (estimated) probability of the passwords "aaa," "bbb" and "ccc" were 3%, 1.7% and 0.8%. After three incorrect login attempts "aaa" (3%), "bbb"(1.7%), and "ccc"(0.8%) are attempted, then Ψ_u is set to $0.055 = 0.03 + 0.017 + 0.008$. We demonstrate the login flow in **Algorithm** 1[1]. Here, u_p denotes the actual password of user u and Ψ (resp. K) are global threshold parameters i.e., the user is locked out if the "hit count" exceeds Ψ (i.e., $\psi_u \ge \Psi$) or if there are too many consecutive mistakes (i.e., $K_u > K$).

Remark: One could consider setting different thresholds (e.g., Ψ_u and K_u) for different users e.g., users with strong passwords might

[1]We omit the description of the password hashing algorithm to ease presentation. Any secure implementation would need to use a salted password hash algorithm with plenty of key-stretching to increase guessing costs for an offline attacker.

be awarded with larger Ψ_u and K_u. However, because Ψ_u and K_u are stored on the authentication server this would leak information about the strength of pwd_u to an offline attacker[2]

Algorithm 1 DALock: Novel Password Distribution Aware Throttling Mechanism

```
Input: Username $u$ and password $p$
1:  function LOGIN($u, p$) 0
2:    if $\psi_u \geq \Psi$ or $k_u \geq K$ then
3:      Reject Login
4:    end if
5:    if $p == u_p$ then
6:      Reset $K_u$
7:      Grant Access
8:    else
9:      $\psi_u \leftarrow \psi_u + CS(p)$
10:     $k_u \leftarrow k_u + 1$
11:     Deny Access
12:   end if
13: end function
```

4.2 DALock **Server Setup**

To implement DALock we need an efficient way to estimate the probability of each guessed password. There are several different ways to accomplish this task. One approach would be to use a password strength meter although they can frequently give inaccurate estimates of true password strength(Jeremiah's Note: *Add citations*). Another naive approach would be to simply 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 [39] proposed the use of 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. We chose to adopt a Differentially Private Count-Median-Sketch. In our experiments we find that the Laplace Noise added to preserve differential privacy does not adversely affect the performance of DALock, and we also found that Differentially Private Count-Median-Sketch outperforms its differentially private counterparts Count-Mean and Count-Min. In the rest of this, we simply refer to Count-Median-Sketch as 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 [39] and/or to help identify IP addresses associated with malicious online attacks [51]. One disadvantage is that the attacker will also be able to view the Count-Sketch data-structure 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 passwords. 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 major risk as most attackers will already know about globally popular passwords e.g., from prior breaches.

Releasing Count Sketch: In this work, we consider the actual usage of a distribution in DALock as releasing it. can potentially acquire knowledge based on the feedback of DALock and therefore it is crucial to ensure privacy protection.

To address such concern (by differential privacy), we assume the server periodically updates the Sketch Count data structure by batching. After a certain period of time, e.g. 180 days, one can apply **Algorithm** 2 initialize a Count Sketch with Laplace noise of magnitude of LAP($\frac{\epsilon}{d+1}$) . After that, one can load the true distribution to the Count Sketch release it.

Algorithm 2.

Algorithm 2 Initializing Count Sketch: Release Count-Sketch using differential private

```
Input: Count Sketch width $w$, depth $d$, users $\mathcal{U}$, privacy
       parameter $\epsilon$,
1:  function SETUP($w, d, \epsilon$)
2:    CS ← new CS(d, w)
3:    for i ∈ CS do
4:      for j ∈ CS[i] do
5:        CS[i][j] = Lap($\frac{d+1}{\epsilon}$)
6:      end for
7:    end for
8:    CS.T += Lap($\frac{d+1}{\epsilon}$)
9:
10:   return CS
11: end function
```

5 MODELING THE ATTACKER

In this section, we discuss possible strategies $\mathcal{A}$ may attempt to perform dictionary attacks guarded by DALock.

In this work, we describe an *unrealistic strong* adversary $\mathcal{A}$ to objectively measure the performance of DALock against dictionary atacks. For any attackers, without knowing system-wide parameters(K, Ψ), and individual parameters (K_u, Ψ_u) at *any time*, it can be infeasible to launch optimal attacks. Consider the following example, $\mathcal{A}$ is trying to crack u's password guarded by $10, 2^{-8}$)-DALock

To objectively simulate a reasonably strong advsersary $\mathcal{A}$, there are several design challenges need to be taken into considerations. Firstly, algorithms described in **Section** 5 assumes $\mathcal{A}$ knows f(p) and CS(p) for each password p. Secondly, both algorithms require $\mathcal{A}$ to know system-wide security parameters K, Ψ, and individual parameters (Ψ_u, K_u) for each user u. Thirdly, it may not be feasible to plan optimal attacks without knowing users' activities in advance. Attempting agreessively with popular ones can lead to throttling due to users' future mistake, on the other hand, adopting conservative approaches can result in low success rate.

Foreseer Adversary: Firstly, we introduce our adversary model $\mathcal{A}$. For each user u, let T = $\{t_1, \ldots, t_n\}$ be timeline for u's login activities. Assume $\mathcal{A}$ has access to an oracle $O(t)$ which takes time

[2]One could potentially avoid storing Ψ_u and K_u 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 such solutions.

t as input, and return the current throttling parameters ψ_t and k_t of u after this login attempt at time t. For the simplicity of discussion, we denote $t - \delta$ to be the moment right before the login attempt at t, consequently, querying $O(t - \delta)$ returns the throttling parameters right before the login. Granting access to O allows $\mathcal{A}$:

- To avoid triggering DALock.
- To be able to plan optimal attack (with expensive computational cost described later).

Based on the setting described in **Algorithm** 1, total attacking budget Ψ decreases over time as u is likely to make more mistakes with respect to the length of time span; however, total attacking budget K can be potentially benefited from a longer time span because traditional counter k resets every time u login successfully. In order to determine the optimal end time t with attack budget Ψ_t and K_t, $\mathcal{A}$ needs to solve the following password knapsack problem for every t ∈ T. By assuming the attack will be terminated at time t, in order to maximize the chance of success:

(1) Find a subset (up to K-1) of passwords S s.t. $f(S)$ is maximized and $CS(S) < \Psi_t$. Where we slightly abuse notation by writing $f(S)$ (resp. $CS(S)$) instead of $\sum_{s \in S} f(s)$ (resp. $\sum_{s \in S} CS(s)$).
(2) Find a password p_{last} for last attempt, after this trial, the user account will be locked if this one is incorrect.

Notice that it is crucial to consider the last attempt, holdout password, specially. Imagine that Ψ is set to be 0.01 while $f(p_1) = 0.02$ and $f(p_2)) = 0.008$. Reserving p_1 for attempt results in 0.028 success rate in lieu of 0.02 for not holdout p_1.

This is a computational challenging task for powerful adversaries who have access to all the above sensitive parameters. In fact, we proved that this is NP-hard by reducing a well-known NP-hard problem subset sum to it in Appendix (**Theorem** .1). Formally speaking, the attacking strategy can be described as the following problem:

*Definition 5.1 (Optimal Strategy - Password Knapsack(*PK*)).* Given a set of passwords P $\{p_1, \ldots, p_n\}$, attacking budget K ,and Ψ. Let $f(p_i)$ be the actual popularity of password p_i, $CS(p_i)$ be the estimated popularity of password p_i. In addition, let s_i and l_i are indicating variables for password p_i, where $s_i = 1$ means password p_i is chosen for the attack and $l_i = 1$ stands for choosing p_i as the holdout password. Clearly, $\mathcal{A}$ wants to find up to K passwords ∈ P s.t.

$$\max \sum_i (s_i + l_i) \cdot f(p_i)$$

subject to,

$$\begin{aligned} \textstyle\sum_i s_i \cdot CS(p_i) &\le \Psi \\ \textstyle\sum_i s_i &\le K - 1 \\ \textstyle\sum_i l_i &\le 1 \\ \forall i \; l_i + s_i &\le 1 \end{aligned}$$

where,

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

s_i and l_i are indicating variables for password p_i. Intuitively speaking, $s_i = 1$ means p_i is selected to be placed in the "knapsack", i.e. to be used for dictionary attack. l_i stands for choosing password p_i as the last attempt. We remark that Password Knapsack problem can be viewed as a two dimensional knapsack problem which does not even admit a PTAS assuming $P \neq NP$[29] (by contrast the regular knapsack problem admits a FPTAS). Since we do not have any known PTAS algorithms for Password Knapsack we consider several heuristic approaches for $\mathcal{A}$ to pick a set S of at most K passwords to guess.

For the former case, we argue it's not feasible for large scale online attack, consider the following realistic setting: $\mathcal{A}$ wants to spend at most 180 days for cracking accounts on small website P with millions of users. $\mathcal{A}$ "magically" gets access to O so he can query the "future" remaining budget on every account at any time. Assume users login into their accounts daily, and let the password dictionary contains 15,000 passwords. If $\mathcal{A}\mathcal{A}$ wants to find the optimal solution, $\mathcal{A}$ needs to solve $180 \cdot 10^6$ instances of PK. Each individual PK instance is likely to have different parameters (K and Ψ budget) due to users' mistakes.

Alternatively, $\mathcal{A}$ can use heruistic approaches to avoid expensive computation. In this work, we consider the following two approaches.

- Dantizig's Algorithm Based[16] (DAB)
- Feasible Most Promising Password First. (FMPPF)

Notice that in every attacks, the most popular password should always be reserved for the final attempt to maximize chance of success unless Ψ is unbounded. Since Ψ threshold is deducted *after* incorrect attempts, attempt the most popular password last allows $\mathcal{A}$ to enjoy attack budget $\Psi' \approx \Psi + \mathrm{SC}(p_1)$.

DAB (**Algorithm** 5, Appendix) sorts passwords $\mathcal{P}_{\tilde{\Pi}} = \{p_1, \ldots, p_n\}$ based the $\frac{f(p)}{SC(p)}$ and select candidates based on the sorted order. Primary incentives of using this algorithm are 1) to take advantage of underestimated passwords and 2) to avoid (severely) overestimated ones.

There are several drawbacks of DAB. Firstly, the progress can be slow because priority are given to significantly underestimated passwords. Intuitively, for popular passwords ($f(p)$ large) the ratio $\frac{f(p)}{sc(p)}$ is likely to be close to 1, therefore, attempts with popular ones are likely to be delayed. Secondly, unlike vanilla version of Knaspack, DAB may not yield a 2-approximation due to the additional constraint on the number of passwords in the knapsack. Third, computation costs are slightly higher for running DABthough both algorithms terminate reasonably quickly.

FMPPF (**Algorithm** 3) uses the default order (sorted based on the actual popularities) and simply selects password p in sorted order s.t.$SC(p) < \Psi$. In short time attack scenarios, FMPPF offers better chance of success than DAB by attempting popular ones first. For long term case, 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** 8.3), the performance of FMPPF is very close to theoretical upper bounds ($\Psi + f(p_1)$).

We found that FMPPF generally performs better than DAB despite of its simplicity. In fact, our simuation shows that FMPPF's performance is close to optimal. Practically speaking, one generally expect $CS(p_i) \approx f(p_i)$ especially when $f(p_i)$ is a popular password. Therefore DAB can hardly gain advantage from popular passwords. Secondly, imagine one bucket passwords by probability ranges, there are plenty of passwords in each bucket. Intuitively, picking

Algorithm 3 FMPPF **Approach**

```
Input: passwords dictionary sorted by actual popularity
       $\mathcal{P}_\Pi = \{p_{\Pi(1)}, \ldots, p_{\Pi(1)}\}$, attack budget $\Psi$ and K.
Return: An array of password sorted in the order of guessing
1: function AttackSetup($\mathcal{P}, f, CS, \Psi, K$)
2:     $S$= []
3:     for $p \in \mathcal{P}_{\tilde{\Pi}}$ do
4:         if CS($S \cup p$) $< \Psi$ and $|S| < K$ then
5:             $S$.add(p)
6:         end if
7:     end for
8:     return Top $K$ of $S$ based on $f(p)$
9: end function
```

passwords ordered by $f(p_i)$ should produce a (almost) optimal solution (quickly).

6 SIMULATING USERS' LOGIN ACTIVITIES

In this part, we focus on discussing how to simulate users' login activities in our experiments. Generally speaking, there are three aspects one needs to consider:

- User's password choice.
- User's login patterns/frequency.
- User's Honest mistake

Simulating Password Choice is a straight forward task. Ideally, one can create users and assign passwords based on aforementioned three datasets. In our simulate, the simulator samples a password based on password distribution. For example, In RockYou, each user has probability 0.89% to have "123456" assigned as his or her password. In each run of simulation, 100,000 users were created in order to have an adequate representation of original distribution. For dataset **LinkedIn** and **Yahoo**, since we don't have the plaintext of passwords, random strings were generated for the purpose of simulation.

Simulating user's login patterns Following prior work we use a Poisson arrival process to model a user's login activities[4][27] with arrival rate parameter t_u for each user u. To verify the performance of DALock over a reasonablly long time span, we simulate the login activities over time span of 180 days, or 4320 hours. In our simulation, each user's login activities can be viewed as a sequence of increasing random variables $0 < T_1 < t_2 < \cdots < 4320 = 180 * 24$. Each random variable T, or login activity, is generated from range 0 to 4320 to represent u' login activities at time R over 180 days time span (rounded to hours). The smaller value of t_u, the more frequent u logins into her accounts. To have a realistic representation of login pattern, for each user u, t_u is sampled uniformly random from { 12, 24, 24 * 3, 24 * 7, 24 * 14, 24 * 30}.

Notice that for each login activity, more than one login attempts can generated because of users' mistakes. Naturally, we simulate each login activity by assuming that users will keep trying until they successfully login into their account, or get locked out unfortunately.

Simulating user's mistakes is the last challenging ingredient for simulating user's login activities. In order to reasonably simulate user's mistakes, two major type mistakes were took into considerations: entering a different password or making typos on the original passwords. We followed a recently published statics on users's mistakes[14] to setup the probability of users' (varies type of) mistakes. Based on the results of the literatures, users have roughly 7.5% chance of making a mistake (Pr(Mistake) = 0.075). Among those mistakes 68% of those incorrect attempts are within editing distance 2. To simplify the model and analysis, we consider that as the probability as making typos. i.e. given a password p, user has probability $0.075 \cdot 0.68$ of typing it wrong. We summarized the distribution of various types of typos in Table 3 for reader's convenience. Further more, we consider the rest 32% errors come from entering wrong passwords(editing distance greater than 2), in another word, entering different passwords. Therefore, each user has probability $0.075 \cdot 0.32$ to select a wrong password to attempt (of course, the user can make typos on top of that!). We setup such secondary passwords (in users' mind) when users are created by randomly sampling a different password for each user based on the distribution of passwords. To help readers capture the essence of user's honest mistakes are simulated in our experiment, we present the flow chart in Figure 3, **Appendix**.

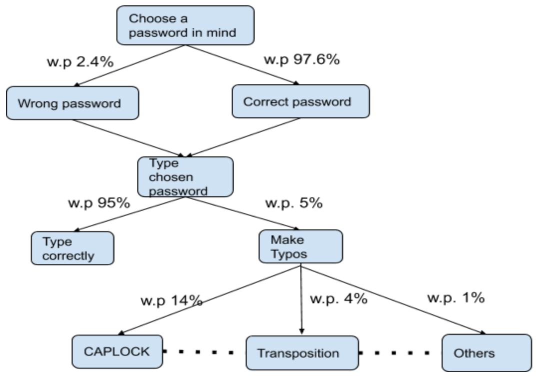

Figure 3: Flow Chart for Simulating Users' mistake

7 SIMULATING ATTACKER

To measure the performance of DALock on dictionary attack, we assume the following reasonably powerful adversary $\mathcal{A}$, or *foreseer*. Briefly speaking, we assume $\mathcal{A}$ is capable of doing the following:

- Knows the exact password distribution.
- Has read access to the count sketch stored on the server.
- Knows security parameters deployed (K and Ψ) on DALock.
- Knows remaining security budgets for each user u.
- Can foresee successfulness of users' login attempts!

The first four assumptions grant $\mathcal{A}$ to be able to potentially setup a profitable order for password cracking based on the knowledges. The final assumption makes $\mathcal{A}$ extremely powerful, as $\mathcal{A}$ can choose whether or not to trigger DALock by smartly planning the attack on each individual user. For example, if $\mathcal{A}$ foresees u will get locked out at time t because of making too many mistakes, $\mathcal{A}$ can arrange multiple guesses before that and choose to lock out the account himself to achieve better profits. On the other hand, if $\mathcal{A}$ notice that u will make some mistakes at time t, $\mathcal{A}$ can arrange less guess attempts before that t so u's account can be (potentially) cracked in the future after threshold reset.

Based on the above discussion, for each user u, given all u' login attempts $\{R_1, R_2, \cdots\} \in 0, 1^*$, $\mathcal{A}$ arrange the attack as follows:

(1) Find the immediate next successful login activity R_t.
(2) If R_t exists
    - Compute the remaining "attacking budget" k_t and ψ_t right before R_t. i.e. compute how much security budget u would have consume by making mistakes.
    - Arrange $k_t - 1$ "high quality" guesses without triggering DALock.
(3) If R_t can not be found: Arrange k_t "high quality" guesses and triggering DALock. This is the end of attack on user u.

8 EXPERIMENTS

In this section, we show the empirical results obtained based on our simulations. We first introduce the general and common setups of all experiments in section 8.1. Then we describe each empirical result and present our analysis on top of it in **section** 8.5

8.1 Experimental Settings

In this part of the section, we focus on discussing the following three key ingredients.

- Experimental Dataset
- Users' activities
- Adversaries' activities

Experimental Dataset We empirically tested DALock on the following three datasets: RockYou[37], LinkedIn[30], and Yahoo[8]. We summarize the characteristics of them in Table 2. For each dataset, we describe it by listing the total number of unique passwords, total number of users' accounts, and frequency of the most common password denoted as $F(p_1)$.

RockYou[37] is a real password corpus contains 14,341,564 unique passwords from a data leakage in 2009. To the best of our knowledge, RockYou is the largest leaked dataset with *unencrypted* passwords. Each line of the file contains the plaintext password, followed by the number of users using that particular password. For example "123456 290729" means290,729 accounts were using "123456" as their passwords.

LinkedIn[30] is an enormous leaked password corpus due to a hack in 2012. It contains more than 60 millions of unique passwords from 174 millions of users accounts. Each line of the file is a two tuple where the first entry corresponding to the number of unique passwords and the second tuple corresponding the number of users who were using each of those passwords. For example "110 392" means there were 110 passwords in the dataset and each of them was used by 392 users.

Yahoo[8] is a sanitized password frequency dataset collected from Yahoo in May 2011. It consists of anonymized password histograms representing almost 70 million Yahoo! users. The format of this dataset is exactly the same as **LinkedIn**.

Dataset	Unique Passwords	Accounts	$f(p_1)$
LinkedIn	60,065,486	174,292,189	0.65%
Yahoo	33,895,873	69,301,337	1.1%
RockYou	14,341,564	32,603,388	0.89%

Table 2: Summary of dataset

8.2 Empirical Measurements of DALock

In this section, we are proud to demonstrate the empirical results obtained from various simulations over long time span (180 simulation days). Briefly speaking, our simulations can be divided into the following two categories

- Security aspect of DALock
- Utility aspect of DALock

All experiments are measured under various combination of security parameters k and ψ, Count-Sketch configuration, we concisely present the following selected but representative settings .

- K: 3, 10
- Ψ: $\infty, 2^{-6}, 2^{-7}, 2^{-8}, 2^{-9}$
- Count Sketch Configuration: d = 5, w = 100,000. ($\approx$ 2 MB space)
- Differential Privacy Budget ϵ: 0.1, 0.5, noise-free

The setting of $K = 3$ corresponding to a conservative threshold used by most websites. The second chosen value, 10, corresponding to the setting recommended by Brostoff et. al [9].

For the second parameter Ψ, setting it to ∞ means password popularity is not used for throttling, therefore it represents traditional k-strikes mechanism. We test the performance of DALock under various settings of Ψ range from 2^{-6}($\approx 1.5\%$) to 2^{-9}($\approx 0.2\%$).

8.3 Security of DALock

In this section, we compare different settings of DALock with traditional K-Strike mechanism. The results can be found in **Figure** 4, **Figure** 5, and **Figure** 6.

Overview The X-axis and Y-axis of each plot correspond to time span over 180 days and the percentage of cracked accounts respectively. Configurations with the same Ψ values are plotted in the same colors. For instance, green for $\Psi = 2^{-6}$. To help readers better intepret the performance of attackers, each plot contains four theoretical upper bound indicators plotted in dash-line. For example, the green straight dash-line stands for the upperbound $\mathcal{A}$ can achieve under the setting of $\psi = 2^{-6}$. Solid and dotted lines correspond to settings of $K = 3$ and $K = 10$ respectively.

Smaller $\Psi \rightarrow$ Better Security

Our results clearly fits the intuition of DALock, adopting smaller Ψ values can significantly penalize the performance of dictionary attacks (under the same K). When large Ψ is deployed, such as 2^{-6} ($> f(p_1)$ on all three datasets), we are not surprised to discover that larger K results in worse security protection. For example, more users are being compromised under the setting of ($K = 10$,$\Psi = 2^{-6}$) than traiditional 3-strike mechanism.

Larger $K \rightarrow$ Worse Security?

Our empirical results verified our conjecture: K has limited impact on throttling when Ψ is properly configured.

Clearly, when Ψ is large, i.e. ∞ and 2^{-6}, $\mathcal{A}$ benefits from larger K. On the first 25 days of attacking, $\mathcal{A}$ is able to crack significantly more accounts on all three datasets under the setting of $K = 10$ and $K = 3$.

Interesting observation can be found when Ψ is properly deployed. On all three datasets, one can observe that as Ψ decreases, the gap between $K = 3$ and $K = 10$ diminishs. Eventually, when Ψ is sufficiently small, i.e. 2^{-8} and 2^{-9}, there is no difference of choosing $K = 3$ or $K = 10$. In those setting $\mathcal{A}$ burns Ψ bugets quickly after few attempts. As a result, in the rest of time, $\mathcal{A}$ is forced to pick less promosing password to utlizie the remaining K budget and result in infinitesimal gain.

Robustness to Laplace Noise We found that Laplace noise introduced by differential privacy has little impact on simulations. Recall **Algorithm** 2, Laplace noise is injected to the total count T and every index [i][j] of Count-Sketch CS .

Firstly, Based on **Table** 2, number of unique passwords are above 10 millions on all three datasets. Therefore, T can hardly diverge significantly from it's ground truth value T'.

In addition, under the setting of dicitonary attacks, AA leans toward attempt popular passwords which have relative high frequency counts. Laplace Noise is also considerably low compare to their ground truth values.

Finally, DALock is also robust to Laplace noise when rare passwords are queried. Recall the fact that Ψ is a fractional counter. As a result, when Laplace noise dominates the ground truth estimation, the maximum error one can have is around $O(\frac{O(\epsilon)}{T})$.

8.4 Usability of DALock

In the previous experiment, we have shown that DALock offers low penetrate rates when Ψ is properly chosen. In this section, we show that one can achieve high usability without compromising security. We quantify utility by counting unwanted lockouts when there is no attacks over 180 days span[3].

Overview The results can be found in **Figure** 7, **Figure** 8, and **Figure** 9. Similarly to last emperiment, each color corresponding to settings. The X-axis and Y-axis of each plot correspond to time span over 180 days and the rate of false positive throttling respectively. Configurations with the same Ψ values are plotted in the same colors.

Larger $K \longrightarrow$ Better Usability Firstly, our results empirically verified the fact that usability can be significantly improved by using the recommended value 10[9]. Based on the plots, one is able to reduce the lockout rates by roughly 80% to 85% simply for adopting $K = 10$ in lieu of 3 when Ψ is fixed. For example, deploying a conservative value $\Psi = 2^{-9}$ results in 3.5% of unwanted locked out when $K = 3$ on LinkdedIn dataset. Switching to $K = 10$ sharply redcue the number of unwantted lock out rates to 0.6%.

$(10, \Psi)$-DALock beats 3-Strike Based on the observation, we found that $(10, \Psi)$-DALock provides strictly better utility compared to 3−stike mechanism. For instance, if the system administor uses DALock with security parameters $\Psi = 2^{-9} (\approx 0.2\%)$ and $K = 10$, less than 1% of the users received throttling. On the otherhand, setting 3−strike mechanism results in roughly 2.7% chance of generating unwanted lockout.

[3]When there is an attack, we believe is reasonable for the server to lock accounts for security purpose

Switching to $(10, \Psi)$-DALock We are thrilled to discover optimistic effects on all three datasets by comparing the usability experiment with the security experiment. Recall that based on **section** 8.3, K barely increases risks when Ψ is properly chosen. To be more specific, $(10, 2^{-8})$ and $(10, 2^{-9})$ DALock system are strictly better than 3-strike mechanism in terms of both security and usability.

Robustness to Laplace Noise Based on our observation, there are two major cases contribute to unwanted throttling. The first case is making too many incorrect attempts which is not influenced by Laplace Noise. The second case, however, is barely inflcuenced by Laplace Noise out of surprise.

8.5 Summary and Discussion

In this section, we summarize our findings over all experiments.

Using differential privacy Based on our results, we conclude it is pratical to use differential privacy version of DALock. Firstly, popularity of popular passwords are unlikely to be distorted due to their magnificant counts. Secondly, it is desirable to misinterpret popularities of rare passwords by $O(\mathsf{Lap}(\frac{d}{\epsilon}))$. Despite the fact that injected noise is much larger, such value is still small compare to Ψ. Finally, one is able to use small privacy budget such as 0.1 to achieve high level protection.

Space Efficient Our experiments demonstrate that one is able to run DALock with a small Count-Sketch for large data. A Count Sketch of size 2MB easily fits into memory and modern cache.

Configuring DALock: Replacing 3-Strike Mechanism Firstly, the experiments clearly demonstate that one shall repalce 3-strike mechanism with a (10,Ψ)-DALock system for better security and usability. For instance, $(10, \Psi = 2^{-8})$-DALock offers more effective defense against dictionary attacks on all three datasets while suffers lower accidental locked out rates.

Since $\Psi + f(p_1)$ naturally represents the maximum damage one can take due to dictionary attacks, it's straight-forward to setup Ψ. Studying exisiting pasword distribution can be helpful for service administors to make the decisions.

9 LIMITATIONS AND DISCUSSION

In this section, we discuss potential limitations of DALock by stepping out of our assumptions and primary focus: dictionary attackes. In addition, we compare DALock with StopGuessing[51] in details to illustrate the fundamental difference of two works.

Other Attacks DALock assumes that adversaries perform rational online dictionary attacks. Therefore, it can be vulnerable against other method such at targetted attack and denial-of-service attacks. Fortunately, one is able to integrate DALock with other defense mechanism such as CAPTCHA to mitigate the issue.

Protecting Weakest User No matter how small Ψ is deplyed, $\mathcal{A}$ can always compromise the weakest users under the general framework of DALock since thresholds are maintained *after* verification.

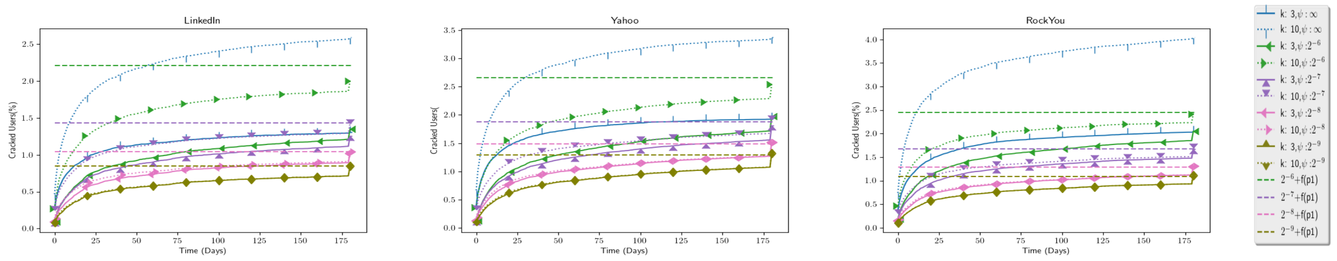

Figure 4: DALock **Without differential privacy**

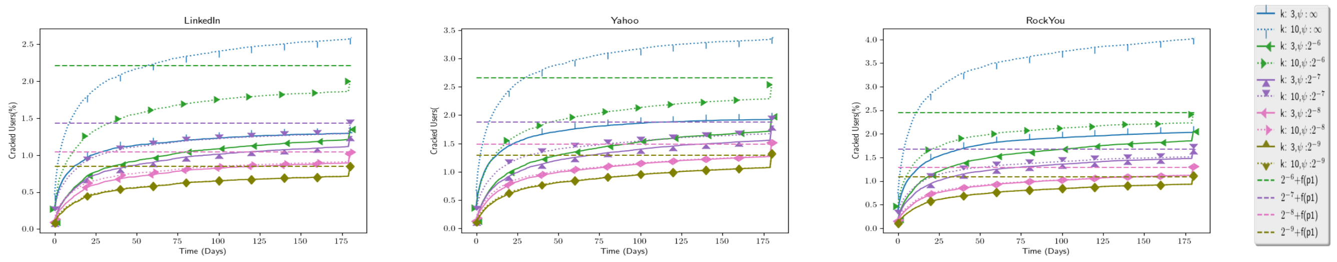

Figure 5: Differential Private DALock **with privacy budget** $\epsilon = 0.1$

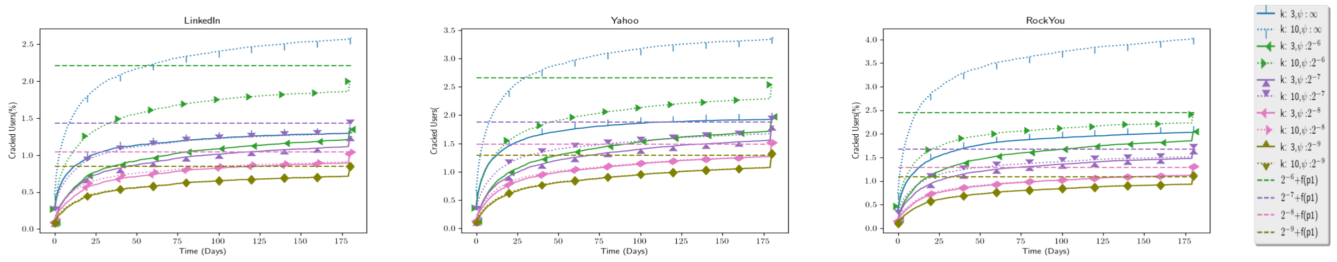

Figure 6: Differential Private DALock **with privacy budget** $\epsilon = 0.5$

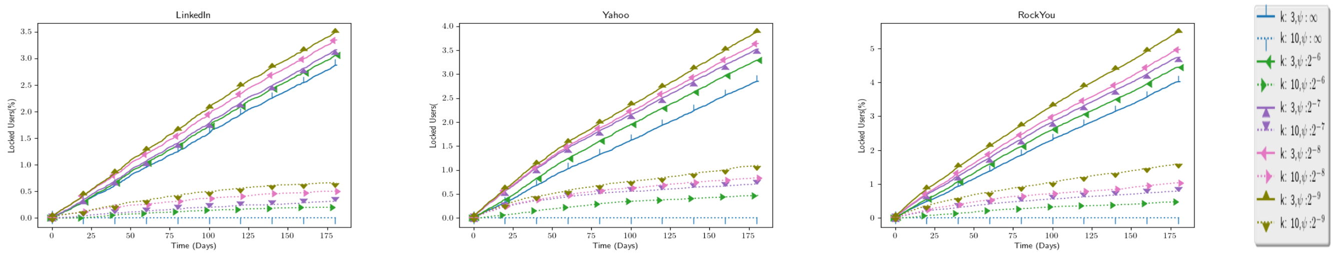

Figure 7: DALock **Without differential privacy**

Without the help of stricter authentication protocols such as 2FA, DALock alone is not able to reduce such lower bound.

Password Distribution DALock assumes that the stored password distribution is close to the real one. Firstly, $\mathcal{A}$ can maliciously register enornmous amount of accounts to lower the popularity of actual frequent passwords. The account creation process shall be carefully auditted to overcome the issue. Secondly, Since users can change their passwords at any time, one should update the Count-Sketch periodically (such as every 6 month).

Simulating Users' mistake In this work, we simulate user's mistakes by considering two types of errors: 1) Typos, and 2) Entering a different password based on the work of Chatterjee et al[14]. There can be a gap between real world scenarios and our simulations.

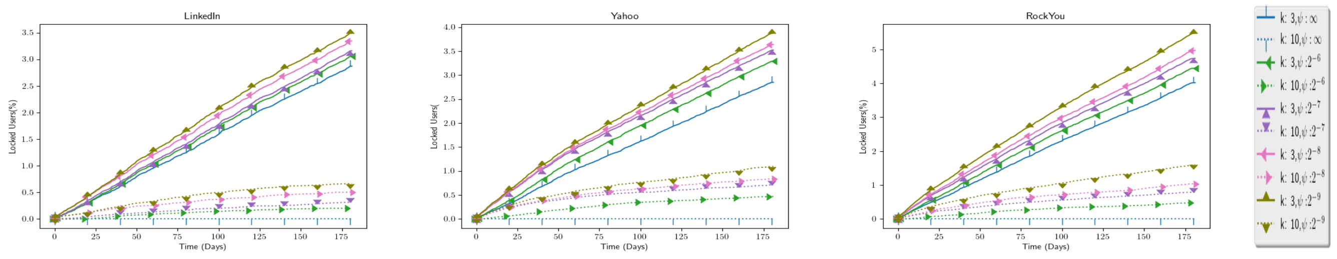

Figure 8: Differential Private DALock **with privacy budget** $\epsilon = 0.1$

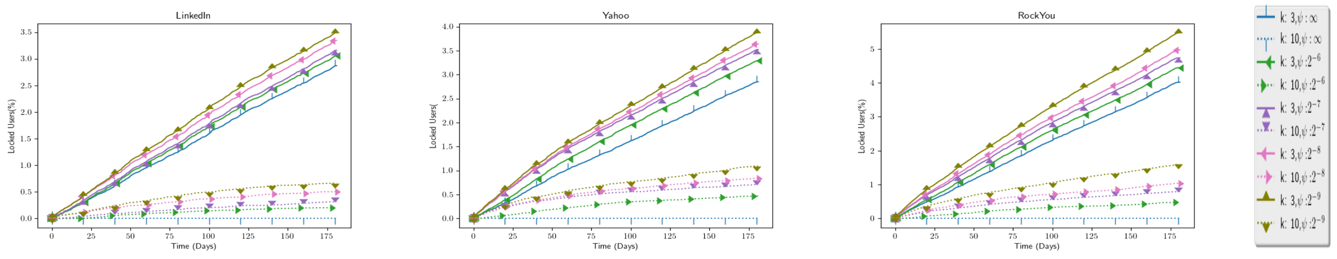

Figure 9: Differential Private DALock **with privacy budget** $\epsilon = 0.5$

For example, users who use malfunctioning keyboards are likely to repeat their mistakes.

(Jeremiah's Note: *A bit awkward to have two comparisons*) **DALock and StopGuessing[51]**

As mentioned in **section** 2, both DALock and StopGuessing utilize the distribution of passwords to defend against dictionary. In this part, we further discuss the difference of these two works.

Different Passwords Distribution Firstly, DALock and StopGuessing collects different distributions. DALock uses true password distriubtion (perturbed by differential privacy) to estimate popularity of password. On the other hand, StopGuessing uses the distribution of *failed attempts*. Maliciously altering the former distribution is a challenging task for attackers because significantly many accounts have to be created. Perturb the distribution stored by StopGuessing is eaiser as it only requires spamming attempts.

Lockout Policy One fundatement difference of DALock and StopGuessing is their lockout policy. DALock locks an *account* if too many *incorrect* attempts are discovered. StopGuessing sliently blocks an *ip address* on a *successful* login after a series of *failed* activities. The later policy can cause a usability concerns if an honest user is silently blocked when they enter the correct password.

Throttling Threshold DALock maintains a *global fractional* hit count threshold Ψ for rate limiting. On contrary, StopGuessing implicitly maintains interger threshold for each user u based on the strength of password p_u. For DALock, service provider can intuitively Ψ based on the risk one is willing to take. For StopGuessing, it is less straight-forward to setup all the parameters such as penality functions.

Against Dictionary Attack DALock and StopGuessing can easily be integrated to provide a solid defense against dicitonary attack. DALock offers concrete account protection because attackers is subject to limited chance of success. This is not true for StopGuessing if the adversary is powerful, e.g. controls a large botnet . Dictionary attcker can attempt real popular passwords while delibrately create‘popular passwords" to circumvent or relief the constraints imposed by StopGuessing.

CONCLUSION

In this work, we purposed a novel throttling mechanism DALock that utilize privately collected passwords distribution to defend against online dictionary attacks. Properly configured DALock offers strictly better protection than 3-strike mechanism while providing significantly more usability.

Algorithm 4 K**-Strike**: Traditional Throttling

```
        Input: Username $u$ and password $p$
 1: function LOGIN($u, p$)
 2:     if k ≥ K then
 3:         Reject Login
 4:     end if
 5:     if $p == u_p$ then
 6:         Reset k
 7:         Grant Access
 8:     else
 9:         $k \leftarrow k + 1$
10:         Deny Access
11:     end if
12: end function
```

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

Algorithm 5 DAB **Attack**

Input: passwords dictionary sorted by their actual popularity over estimated popularity $\mathcal{P}_{\tilde{\Pi}} = \{p_{\tilde{\Pi}(1)}, \ldots, p_{\tilde{\Pi}(n)}\}$, attack budget Ψ and K.

Return: Up to K passwords

```
1:  function DAB ATTACK($\mathcal{P}_{\tilde{\Pi}}, \Psi, K$)
2:    $S$= []
3:    while $S$ changes do
4:      for $p \in \mathcal{P}_{\tilde{\Pi}}$ do
5:        if CS(p) > $\Psi$ then
6:          continue
7:        end if
8:        if CS($S \cup p$) < $\Psi$ and $|S| \leq$ K then
9:          $S$.add(p)
10:       else if $f(p) > f(S)$ then
11:         $S \leftarrow$ {p}
12:       end if
13:     end for
14:   end while
15:   return Top $K$ of $S$ based on $f(p)$
16: end function
```

Typo Types	Chance of Mistake(Rounded %)
CapLock On	14
Shift First Char	4
One Extra Insertion	12
One Extra Deletion	12
One Char Replacement	31
Transposition	4
Two Deletion	3
Two Insertion	3
Two Replacement	10
Others	8

Table 3: Typo Distributions[14]

Proof: We first formally define 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. $\sum_{i \in S} x_i = T$?

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 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.

Note that in the new ACM style, the Appendices come before the References.

REFERENCES

[1] Fadi Aloul, Syed Zahidi, and Wassim El-Hajj. 2009. Two factor authentication using mobile phones. In *2009 IEEE/ACS International Conference on Computer Systems and Applications*. IEEE, 641–644.

[2] Apple. Apple Differential Privacy Technical Overview. (????). https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf Retrieved 25, Apr. 2019.

[3] Apple. Learning with Privacy at Scale. (????). https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html Retrieved 25, Apr. 2019.

[4] Jeremiah Blocki, Manuel Blum, and Anupam Datta. 2013. Naturally Rehearsing Passwords. In *Advances in Cryptology – ASIACRYPT 2013, Part II (Lecture Notes in Computer Science)*, Kazue Sako and Palash Sarkar (Eds.), Vol. 8270. Springer, Heidelberg, Germany, Bengalore, India, 361–380. https://doi.org/10.1007/978-3-642-42045-0_19

[5] 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.

[6] 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

[7] 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.

[8] 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

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

[10] 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

[11] 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

[12] 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)*, Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo (Eds.), Vol. 2380. Springer, Heidelberg, Germany, Malaga, Spain, 693–703. https://doi.org/10.1007/3-540-45465-9_59

[13] 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

[14] 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

[15] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

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

[17] digital guardian 2018. Uncovering Password Habits: Are Users' Password Security Habits Improving? (2018). https://digitalguardian.com/blog/uncovering-password-habits-are-users-password-security-habits-improving-infographic

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

[19] 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

[20] 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.

[21] 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.

[22] Haichang Gao, Jeff Yan, Fang Cao, Zhengya Zhang, Lei Lei, Mengyun Tang, Ping Zhang, Xin Zhou, Xuqin Wang, and Jiawei Li. 2016. A Simple Generic Attack on Text Captchas. In *ISOC Network and Distributed System Security Symposium – NDSS 2016*. The Internet Society, San Diego, CA, USA.

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

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

[25] Ariel Gordon and Richard Allen Lundeen. 2014. Efficiently throttling user authentication. (Nov. 25 2014). US Patent 8,898,752.

[26] Have I Been Pwned 2019. Have I Been Pwned. (2019). https://haveibeenpwned.com

[27] 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

[28] 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.

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

[30] LinkedIn n.d.. LinkedIn Password Corpus. (n.d.). https://hashes.org/public.php

[31] 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.

[32] 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

[33] Arvind Narayanan and Vitaly Shmatikov. 2006. How to break anonymity of the netflix prize dataset. *arXiv preprint cs/0610105* (2006).

[34] Arvind Narayanan and Vitaly Shmatikov. 2008. Robust de-anonymization of large datasets (how to break anonymity of the Netflix prize dataset). *University of Texas at Austin* (2008).

[35] Thu Pham. 2019. STOP THE PWNAGE: 81% OF HACKING INCIDENTS USED STOLEN OR WEAK PASSWORDS. (2019). https://duo.com/decipher/stop-the-pwnage-81-of-hacking-incidents-used-stolen-or-weak-passwords Retrieved January 21, 2020.

[36] 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

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

[38] Ravi Sandhu, Colin Desa, and Karuna Ganesan. 2005. System and method for password throttling. (April 19 2005). US Patent 6,883,095.

[39] 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.

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

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

[42] Andreas Tuerk. 2019. To stay secure online, Password Checkup has your back. (2019). https://www.blog.google/technology/safety-security/password-checkup/

[43] Liam Tung. 2019. Ransomware crooks hit Synology NAS devices with brute-force password attacks. (2019). https://www.zdnet.com/article/ransomware-crooks-hit-synology-nas-devices-with-brute-force-password-attacks/ Retrieved January 21, 2020.

[44] 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)*, Eli Biham (Ed.), Vol. 2656. Springer, Heidelberg, Germany, Warsaw, Poland, 294–311. https://doi.org/10.1007/3-540-39200-9_18

[45] 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.

[46] 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.

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

[48] 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)*, Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows (Eds.), Vol. 9878. Springer, Heidelberg, Germany, Heraklion, Greece, 111–131. https://doi.org/10.1007/978-3-319-45744-4_6

[49] 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.

[50] 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

[51] Stuart Schechter Yuan Tian, Cormac Herley. 2019. StopGuessing: Using Guessed Passwords to Thwart Online Guessing. In *4th IEEE European Symposium on Security and Privacy*. IEEE.

[52] ZDNet 2019. Facebook passwords are not case sensitive. (2019). https://www.zdnet.com/article/facebook-passwords-are-not-case-sensitive-update/