Improving the Systematic Generation of Secure and Memorable Passphrases by MASCARA

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

by

Kousshik Raj M.
(Roll No. 17CS30022)

Under the guidance of
Dr. Mainack Mondal

Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
West Bengal, India
November, 2021

Certificate

*This is to certify that the work contained in this thesis titled "**Improving the Systematic Generation of Secure and Memorable Passphrases by MASCARA**" is a bonafide work of **Kousshik Raj M. (Roll no. 17CS30022)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur under my supervision and that it has not been submitted elsewhere for a degree.*

Dr. Mainack Mondal
Associate Professor
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur,
West Bengal

November, 2021
Kharagpur

Acknowledgements

It is my bound duty and moral responsibility to offer my deepest gratitude to the Lord Almighty, who has been my strength in times of weakness and hope in times of despair.

Next, I would like to express my sincerest gratitude to my guide, Dr. Mainack Mondal, for his exceptional guidance and support, without which this project would not have been successful. He has always pointed me in the right direction whenever I am lost and unsure how to proceed further. I also want to extend my heartfelt thanks to Prof. Niloy Ganguly for his valuable ideas and suggestions, all of which had a huge impact in the way the work proceeded.

Furthermore, I would also like to thank my fellow students Avirup Mukherjee and Shivam Kumar Jha, as the work came out as a success only through the valuable efforts they contributed. Many thanks to everyone who made this work a successful endeavour.

Kousshik Raj M.

Abstract

Passwords are the most common mechanism for authenticating users online. However, prior studies have established that it is inherently difficult for users to create and manage secure passwords. Passphrases (human-readable phrases consisting of letters and numbers) are often recommended as a usable alternative to passwords, which would potentially be easy to remember for users and hard to guess for attackers. In fact, passphrases are the only go to in many cases where security is deemed extremely important like master secrets for password managers, crypto-currency wallets. Furthermore, passphrases can be used as a context to generate more secure passwords, and if the passphrase is memorable, so is the password generated.

However, user-chosen passphrases from the wild, fall short of being secure enough to be used in sensitive cases, as they tend to be biased towards users' likes. On the other end of the spectrum, machine-generated passphrases are difficult to remember, as they usually tend to be meaningless combination of words, even though they are very secure. To that end, Mascara was proposed. Mascara is a systematic approach for the generation of secure yet memorable passphrases. The core of Mascara is to compromise for a reasonable trade-off between security and memorability.

The system of Mascara also includes a framework for quantifying the measures of memorability (how easy is to remember the passphrase is) and guessability (how secure the passphrase is). With the help of the above frameworks and the use of constrained Markov generation, passphrases are generated while simultaneously optimising memorability and guessability. The various evaluation performed indicated a significant improvement in terms of the existing models for generating passphrases while keeping in mind the trade-off between guessability and memorability.

The work in this thesis aims to improve upon the existing Mascara system. We primarily focus on three areas: 1) Improving the generation time of passphrase, which has huge scope for improvement 2) Automating the finding of the optimal parameters for the constrained generation as opposed to the initial manual try and test approach 3) Analysing one of the leading memorable passphrase generators [1] and comparing its performance against Mascara.

The improvements not only increased the overall performance of the system, but it also provided an insight of how Mascara compares to some of the state-of-the-art passphrase generation targeting memorability.

Contents

Chapter 1

Introduction

Passwords are the most common mechanism to authenticate users on the Internet. However, despite the number of users flocking towards passwords and passphrases, the generation and management of the same are still a problem even today. This is primarily because users are not good in using passwords and passphrases that are strong and hard to guess but also easy to remember and enter.

Towards generating strong and memorable passwords, we aim to leverage the advantage of generating suitable passphrase and then convert them to passwords according to our standards using the generated passphrases as context [2] (e.g. using the starting and ending letters of each word with suitable alphanumeric value as conjunctions that join them, etc.). In fact, in many scenarios like securing crypto-currency wallets [3], password managers [4], ssh-keys [5], etc. passphrases are strictly recommended over generic passwords because of the advantage they have over memorability and guessability (passphrases can be arbitrarily wrong and can still be memorable).

Some of the common and standard methods of generating passphrases till now has been the algorithms that are variants of Diceware [6, 7] - a method where you choose random words from a given word list (vocabulary) and combine them into sentences that don't really make sense. However, Shay et al. [7] have shown that passphrases generated by this approach are error-prone for the users and are not easily memorable as it just resembles the memorability of randomly generated passphrases. On the other extreme, user generated meaningful passphrases are easy to remember, but they are easy to crack as well, thus making them less secure [7, 8].

Mascara is a system proposed to approach the above problems. It primarily tries to address the question of whether it is possible to create usable passphrases which are easy to remember for users yet hard to guess for an adversary. The system includes a framework for quantifying the measures of memorability and guessability. With these frameworks along with the use of constrained Markov Generation trained on a processed dataset of commonly used words by the user, Mascara generates passphrases that are memorable while also being secure. The evaluation of the system and comparison with various benchmarks highlights its performance when it competes with established methods.

But, we still believe this could be further improved upon. First, the current system tries its best to ensure that the generated passphrase is a valid phrase linguistically. This goes a long way to increase memorability on the compromise of a small amount of guessability. But, another factor also has to be considered, which is the generation time for a passphrase. Ensuring a linguistically correct passphrase results in hundred times more rejections than not doing so. In fact, it is not necessary that a sentence has to be linguistically meaningful for it to be memorable. Along with the addition of some heuristic in the generation, we can guarantee an increase in guessability, without compromising much on memorability all the while generating passphrases hundred times faster.

Secondly, the parameters involved in the constrained Markov generation haven't been tuned properly. It was manually set after trying and testing over some range of values, without any logical comparison relation when two measures are involved - guessability and memorability. In this situation, a new evaluation metric was introduced that enables us to perform a grid search on the range of the parameters. This has significantly improved the performance of the model.

Finally, although the benchmarks used in evaluation for the system are really good and standard approaches and algorithms for passphrase generation, one of the most famouss and successful algorithm, *ReadablePassphrases* [1], wasn't considered. As the algorithm targets the generation of readable and memorable passphrases, it can be considered one of the best benchmarks available out there. Thus, we analyse this algorithm, calculate guessranks and memorability, and show how Mascara performs against this approach.

1.1 Objective

The main objective of this work is to propose various enhancements to the system of Mascara, which aims to generate memorable and secure passphrase using constrained Markov generation along with the help of the frameworks defined for memorability and guessability.

- Improve the passphrase generation time by removing redundant linguistic checks and introducing fast and robust heuristics in its place.

- Solve the problem of tuning the parameters in the Markov process. Establish a metric that involves the guessability and memorability of the generated passphrases so that the grid search for the parameters could be carried out without manual intervention.
- Analyse one of the most popular approach for generating passphrases that are memorable from the Internet and introduce it as a benchmark in the evaluation of Mascara passphrases.

Improving all these and with some other minor adjustments, it could potentially boost the performance of the system drastically, as will be shown later. The rest of the thesis is organized as follows: First, we will have a look at the background and works related to passphrase generation in Chapter 2. Then we will move on to chapter 3, where we introduce the Mascara system and other related essentials. In chapter 4, we present the work carried out and followed by describing the experiments carried out and their results in chapter 5. Finally, chapter 6 will conclude our work, summarizing what we have seen so far and discuss the possible future directions for this work.

Chapter 2

Background and related work

We will first provide some background on passphrases as an important authentication mechanism. Then we will discuss the generation as well as measures for security and memorability of authentication mechanisms like passphrases. Finally, we discuss the threat model used by Mascara as well as in the work that follows.

2.1 Passphrases as Authentication Mechanism

Many earlier works focused on passwords as an authentication mechanism for critical as well as non-critical infrastructures [9]. Unfortunately, earlier work also demonstrated that users tend to often choose predictable passwords and reuse the same passwords across multiple accounts [10, 11]. Consequently, multiple mechanisms are proposed to improve the security of authentication mechanisms, ranging from designing better password meters to creating mnemonic passwords [12].

To that end, in the past decade passphrases are being put forward as an alternative or complementary mechanism for passwords. National Institute of Standards and Technology (NIST) defined a passphrase to be a *memorized secret consisting of a sequence of words or other text that a claimant uses to authenticate their identity* [13]. Intuitively, passphrases are likely to be easier to remember than passwords (due to their closeness to natural language) for users as well as harder to guess (due to their length) for adversaries. In fact, Shay et al. [7] demonstrated (using a user study) that even simply using passphrases comprising of three to four words can be comparable in terms of entropy with passwords generated using more-involved methods while also accounting for memorability, which underlines the utility of good passphrases.

We note that the earlier work on password creation often focused on generating secure and memorable passwords over the years. They used techniques like contextual cues, portmanteau, or mnemonic based generation [14, 15, 12, 16]. These techniques to improve memorability often aim to associate a context (like a memorable phrase) associated with the password. Consequently, our work on secure and memorable passphrase generation is complementary to the generation of memorable passwords - a secure and memorable passphrase can easily be leveraged to create a secure password (while using the passphrase as context for memorability).

However, earlier studies have not systematically investigated the guessability-memorability trade-off for passphrase generation and check how to create passphrases that are easy to remember for users and hard to guess for attackers. The Mascara system formulates the memorability-guessability trade-off problem, analyze user-generated passphrases from the wild, and design an algorithm to generate memorable yet hard-to-guess passphrases.

2.2 Generating Passphrases

Today, generating passphrases often involve selecting random words from an English wordlist. For example, Diceware is one popular approach that chooses a random set of pre-determined number of words from a wordlist and concatenate them to create a passphrase [6]. Many other passphrase generation techniques are just variation of this approach with changing the wordlist from service-to-service [17]. A similar web tool, produces readable passphrase [1], a concept similar to diceware, on predefined syntactic templates.

In fact, a recent work even proposed to enable users to choose the words [18]. However, this same work also noticed that users tend to choose more frequently used words, making the passphrases less secure. Consequently, even though secure and memorable passphrases are useful (both as a standalone authentication mechanism, as well as for creating strong passwords), there is a dearth of such passphrase generation methods. Either memorability of today's passphrases suffer from an opaque system-provided choice of wordlists or their security suffer from users choosing common phrases and words (affecting guessability) [8]. Mascara works on this problem and systematically strike a balance between two contradicting dimensions of guessability and memorability of passphrases.

2.3 Measuring Security and memorability

Measuring the security (i.e., how easy is it for an attacker to guess) is a well studied topic [19] for passwords. Such earlier work considered different guessing scenarios based on data available to the users or using Neural Networks for guessability estimation [20, 21, 22, 23, 24, 25]. However, there is relatively little similar work in the domain of passphrases.

2.3.1 Measuring Security

Earlier works found that the guessability of passphrases can be decreased (making them more secure) by increasing the entropy via either semantic noises or increasing wordlist size [26]. However, if the users are given control, they generally tend to introduce bias and opt for common phrases, reducing security [12]. In all of these works, the security of passphrases is generally measured through either user-based surveys or via entropy [27]. Building upon this earlier work and earlier work on password meters [22, 23, 24, 25], Mascara defines a *guessrank* metric for measuring guessability of passphrases in their setup. Specifically, Mascara estimates a guessrank for each passphrase - higher the guessrank, higher the security.

2.3.2 Measuring Memorability

Previous works tried to correlate password memorability with frequency of passwords, login durations, and even keyboard pattern [28, 29]. Some other works used methods like encoding random n-bit strings for generating memorable passwords or using chunks [11, 30]. All of these cases measured memorability of passwords based on user surveys instead of an automated linguistic metric [31, 32, 15, 7]. Although useful, these works considering memorability of passwords are complementary from that of passphrases. Passphrases often contain possible linguistic properties (the order in which words are presented) which is generally absent in password. To that end, there is some work on the memorability of English phrases. For example, work by Danescu et al. [33] has tried to measure memorability of popular movie quotes using lexical distinctiveness. Some Human Computer Interaction studies identified Character Error Rate (CER) as measure for memorabilty of passphrases [34, 35]. Mascara builds on this research, identify underlying factors affecting memorability of phrases, and consequently optimize to improve memorability of the generated passphrases.

As can be seen, there is not much work on systematically generating passphrases while balancing memorability and security. Mascara tries to bridge this gap.

2.4 Threat Model

Mascara considers a powerful offline generalized untargeted adversary. In the threat model assumed, the adversary knows the exact corpus from which our systems are generating passphrases and the exact algorithm of passphrase generation. However, the adversary does not have knowledge of any of the randomized components of markov models and they do not target a particular user or group of users.

Drawing from earlier work, adversarial guessing was used as an estimate of passphrase strength [36, 31] against this adversary. Thus, the primary challenge for the attacker is to generate an ordered list of passphrase guesses ω_1, $\omega_2..\omega_n$ to get to the target passphrase ω as early as possible (least number of guesses). Since a probability of generation was obtained, it can then be utilized to estimate a guess rank β for a given passphrase. Note that, in a practical setting of online attack, the adversary is likely to be severely bounded by rate limiting and is likely to not know the exact system parameters. Hence the results presented in Mascara, as well as in this work, are a lower bound of the number of guesses needed for an adversary to guess the passphrase.

Chapter 3

MASCARA

In this chapter, we will present a brief overview of the memorability and guessability framework proposed by Mascara, and then we move on to see how the system approaches the generation of secure and memorable process.

3.1 Essentials

To overcome the limitations of user-generated passphrases (easy to guess) and machine generated random passphrases (hard to remember), Mascara was designed, that can generate passphrases that are memorable yet hard to guess. To do so, Mascara uses a constrained Markov generative process. The constraints are based on the approximation of the memorability and guessability metric that is defined. First, the training data and the training of the Markov model is described.

3.1.1 Training Corpus

The unigram and bigram Markov models that are required for training Mascara are extracted from Wikipedia as a corpus of human generated text data. A recent dump of Wikipedia articles [37] was collected and its contents were extracted based on the titles of the articles. Only the top 5% articles based on the page view count, aggregated over the last five years, are considered. The granularity of the views were set to be monthly. This only strengthens the scope of the corpus, without having to use a huge dataset, to cover all possible domains.

The raw Wikipedia text was cleaned by removing all tags, URL links, and captions using a number of regular expressions. The words were then normalized to lower case as only lower case words are considered in the passphrase generation. in our passphrase (such passphrases are also easy to type [38]). Words with less than three characters, as well as numeric or alphanumeric words, were removed. Finally, a textual corpus comprising of 3.3 million sentences were obtained (delimited by a period '.', '!', or '?'), from 8,210 Wikipedia articles, containing a total of 2,95,02,034 words (and 4,55,614 unique words).

This dataset (called Wiki-5) is used as a universal corpus for CER and guessrank estimation throughout Mascara, as well as our modified version. This is also used for the passphrase generation.

3.1.2 Training Markov Models

The unigram and bigram models are trained on the Wiki-5 data. To do so, a <s> token is added at the beginning of every sentence and an <e> token at the end of every sentence identified in the dataset. Then all bigrams are constructed and stored in a bi-level dictionary counting the number of times they appear in the dataset. The words were used as it is without performing any stemming or lemmatization.

This model will be referred to as $\mathcal{M}$ henceforth, and the log probabilities of unigram and bigrams in the dataset as L_1 and L_2. Thus $L_1(w) = \log(\frac{f_w}{\sum_w f_w})$ and $L_2(w, w') = \log(\frac{f_{ww'}}{\sum_{ww'} f_{ww'}})$, where f_w and $f_{ww'}$ denote the frequency of the of the word w and the bigram ww', respectively. All logarithms are over base 10. Also, $\mathcal{M}.\mathsf{next}(w)$ is assumed to be a function that returns all the words that appear after w in the corpus.

3.1.3 Estimating Memorability

Quantitatively measuring memorability of passphrases is difficult. Leiva et al. [39], via a news-text entry task, showed that character error rate (CER) can be used as a proxy for memorability. CER measures the rate of error per character while typing a text. Thus, CER is a good enough measure to quantify memorability.

Leiva et al. also give a model to predict the CER of a text. Inspired by their approach, the following equation provided in [39] can be used to measure CER of a passphrase s as $\text{CER}(s) = c_1 \cdot L_1(s) + c_2 \cdot L_2(s) + c_3 \cdot \sigma_{\text{chr}}(s)$. Here, L_1 and L_2 denote the log of the probabilities of the passphrase s according to a unigram model and a bigram model, and σ_{chr} denote the standard deviation in the number of characters in each word in the passphrase; c_i denotes parameters that are learned from the training data. Note, Leiva et al. did not consider bigram model probabilities, however, for passphrases, it is hypothesized that bigram model will be important, as it will help maintain syntactic structure, or in better words, "lexical distinctiveness" to increase memorability, as also pointed out in [33].

The CER estimation is retrained based on the Markov models on the dataset used by Leiva et al. [39]. The retrained new model yields a good fit ($R^2 = 0.58$) for the CER estimate. The values obtained for c_1, c_2, and c_3 in the equation are $\mathrm{CER}(s) = -0.0342 \cdot L_1(s) - 6.46 \times 10^{-3} \cdot L_2(s) + 1.19 \times 10^{-4} \cdot \sigma_{\mathrm{chr}}(s)$.

3.1.4 Estimating Guessability

A good passphrase must not be easy to guess. Guessability of passwords (and passphrases) is measured based on their guess rank according to the attacker's understanding of the passphrase distribution [40]. Passphrases with higher probability has lower guess rank - higher guessability. Here, it is assumed that the attacker estimates the probability of passphrases according to a bigram Markov model. This is reasonable as our generative method also uses a bigram Markov model albeit with some constraints.

The probability of a passphrase $s = w_0 w_1 \ldots w_{n+1}$ according the Markov model $\mathcal{M}$ that was trained is $\Pi_{i=0}^{n} L_2(w_i w_{i+1})$. Here, w_0 = <s> and w_{n+1} = <e>. Estimating guess rank can be very expensive as often they require explicit enumeration of billions of passphrases. Dell'Amico et al. [41] proposed a Monte-Carlo method to estimate guess rank. Mascara also measures the strength of a passphrase generated by our setting based on their guess rank according to a distribution. This works better than entropy-based approaches to distinguish between passphrases of similar lengths.

3.2 Generative Model

Here, Mascara aims to generate passphrases such that they follow some syntactic structure so that they can be memorable, but ensure that they are harder to guess by having low probable bigrams. It carries this out via constrained Markov generation approach as we describe below.

3.2.1 Constrained Generation

The passphrases in Mascara are generated incrementally starting with a start symbol <s>. At each stage a word is selected such that it satisfies the following constraints according to the score function derived. The score function is modeled on the observation that CER can be computed incrementally. That is given a partially generated passphrase $s_i = w_1 \ldots w_i$, one can compute the intermediate CER value using the following equation.

$$S(w_1 \ldots w_i) = c_1 L_1(w_i) + c_2 L_2(w_{i-1}, w_i) + c_3 \sigma_{\mathrm{chr}}(w_1 \ldots w_i)$$

Similarly, the guess rank is also estimated using the bigram probability, L_2. Thus the following constraints are imposed while generating passphrase:

- $S(w \dots w_i) \leq \theta_1$ and
- $L_2(w_{i-1}, w_i) \leq \theta_2$

where θ_1 and θ_2 are two parameters of the system. At every step of the generation, a word is randomly selected such that these two constrains must be satisfied by the intermediate passphrases. Because the score S can be computed incrementally, we remove the words not satisfying this constraint from the support before sampling in a step.

The thresholds provide control over the generated passphrases on whether it should be closer to English text with some semantic meaning (making them easy to remember and easy to guess), or devoid of semantic meaning (harder to guess but also harder to remember). Here, a compromise is made to find passphrases that maintain the syntatic structure while being semantically meaningless.

3.2.2 The Final Algorithm

The Mascara algorithm, which takes a step-by-step approach to generate passphrases under constraints of memorability and guessability while trying to preserve its syntax and meaning, is presented in Figure 3.1.

The approach is greedy, and not optimal. But, replacing the corpus or changing any of the variables can essentially just be a straight swap with the existing one, based on user preference or need, thus making the algorithmic approach a generalized version of generating optimized passphrases step wise, based on the constraints.

The choice of θ_1 and θ_2 are very important. Mascara tries to ensure that, the algorithm favors rare bigrams (low θ_2) while maintaining a low intermediate CER score (low θ_1). Keeping this in mind, a suitable value for θ_1 and θ_2 is set after manually trying out a lot of combinations and checking the quality of generated passphrases. For the generation method, Mascara sets θ_2 to be at least 80% of the maximum possible value of L_2 and θ_1 to be 50% of the maximum possible value of L_1. This will ensure that the generated passphrases contain rare bigrams and thereby high guess rank.

The generation method of Mascara is incremental ensuring the invariant of CER (memorability) and guess rank (guessability). An alternative approach would have been generating the whole passphrase of length l, and then checking if the CER and guess rank constraints are met. Although this will ensure better results, the tradeoff in time is nowhere near worth it.

```
GetFirstWord($\mathcal{M}$) :
$W \leftarrow \mathcal{M}.\mathsf{next}(\mathsf{<s>}) \setminus \mathsf{B}$
$w \leftarrow_{L_1} W$
return $w$

MascaraGen($l, \mathcal{M}$):
$w_1 \leftarrow \mathsf{GetFirstWord}(\mathcal{M})$
$i \leftarrow 2$
while $i \leq l$ do
    $W' \leftarrow \mathcal{M}.\mathsf{next}(w_{i-1})$
    if $i \in \{1, l\}$ then $W' \leftarrow W' \setminus B$  /* Remove stopwords */
    if $i \geq 2$ then  /* CER and Guess rank constraint */
        $W' \leftarrow \{w \in W' \mid S(w_1 \dots w) \leq \theta_1$ and $L_2(w_{i-1}, w) \leq \theta_2\}$
    if $i = l$ then  /* Ends in a end symbol */
        $W' \leftarrow \{w \in W' \mid \mathsf{<e>} \in \mathcal{M}.\mathsf{next(w)}\}$
    $w_i \leftarrow_{\$} W'$
    if $w_i = \perp$ or $i = l$ then
        $W \leftarrow \mathcal{M}.\mathsf{next}(\mathsf{<s>}) \setminus \mathsf{B}$  /* No passphrase found; restart */
        $w \leftarrow_{L_1} W$
    else  $i \leftarrow i + 1$
return $w_1 \dots w_l$
```

Figure 3.1: *Mascara's constraint generation algorithm. The algorithm generates a passphrase of length l given a bigram Markov model $\mathcal{M}$. Here B is a set of stop words, and* <s> *and* <e> *are the start and end symbols used in the Markov model. Furthermore, $\leftarrow_{L_1} W$ denotes sampling from the support W but according to the probability distribution assigned by unigram probabilities (without log); similarly $\leftarrow_{\$} W$ denote sampling uniformly randomly from the elements in W.*

Chapter 4

Proposed Work

As we have discussed before, the goal of this work is to improve the performance of the system as well as the evaluation approach in Mascara. We do this in a broad range, starting from removing redundancies to introducing new algorithms for comparison. In this chapter, we delve into detail the various improvements proposed, and the results of which will be presented in the next chapter.

The first section shows how the generation of passphrases were sped up. Next, we present an approach to optimise the search of optimal parameters in constrained Markov Generation. Finally, we involve one of the most popular tools for generating readable passphrases and compare the performance of Mascara with it.

4.1 Improving Generation Time

As we have seen earlier, Mascara uses a constrained incremental generation to generate passphrases, in which all the intermediate passphrases must stick to the constraints imposed on the CER and Guessrank (indirectly). This ensures that the final CER of the passphrase, although not the best, is reasonably good. But a good CER doesn't really imply that the generated passphrase makes sense linguistically. For example, *adolf took just missing south spoke the* is one such example passphrase that was generated.

4.1.1 Before Modification

As people associate memorability with meaningful sentences, Mascara also makes an approach to make the sentences as meaningful as it can, albeit in a very straightforward way. Instead of modifying the generation process itself, the approach was to make a wrapper over the system so that only passphrases generated that are meaningful are considered, and the rest are rejected. This means that, to generate a passphrase, Mascara algorithm is repeatedly called until a meaningful passphrase is generated. To check whether a sentence is meaningful or not, *PhraseMachine* [42] library is used.

A quick reader might have already noticed the problem with this approach. The fraction of meaningful sentences generated through the Mascara model (without the wrapper), when we use only bigram probabilities and CER values is not going to be high. In fact, it is going to be very low. After a quick analysis, it turned out that more than 96% of the generated passphrases were rejected after applying the wrapper, even though most of the generated passphrases 'almost' make sense. As a consequence, the generation time for passphrases scales up dramatically (obviously!). Although this effect is not much when we generate 1 or 2 passphrases, when we want number of passphrases in the range of 100 to 200, the user is going to be disappointed.

4.1.2 Removing Redundancy

To do something about this, we compared the generated passphrases before and after applying the wrapper, and we noticed something significant. There is a stark decrease in the guessrank of the generated passphrases, while the CER values did not see any significant increase after applying the wrapper. Why is this important? As we have used the CER values to quantify memorability, if we are slowing down the system by nearly 25 times, we expect to see some improvement in the results. Although we can claim a qualitative improvement, we are not really able to quantify that.

Thus, the use of *PhraseMachine* is pretty much redundant in the system, and can be removed. But, to make sure that the generated passphrases are not completely out of context, we ensure that the passphrase doesn't start with or end with a stopword. More heuristics can also be applied during the intermediate passphrase generation (like ensuring adjacent words follow PoS rules) that can ensure meaningfulness of the generated passphrase. The reason the use of *PhraseMachine* is costly because the rejection takes place at the very end, because of which a rejection results in the restart of the process. The same couldn't be said about using intermediate heuristics.

4.2 Optimising Parameter Tuning

We have previously seen that Mascara relies on the constrained Markov generation to produce passphrases with satisfactory CER and Guessrank values. It does so with the help of the parameters θ_1 (imposes constraint on CER) and θ_2 (imposes constraint on bigram probabilities, thereby restricting Guessrank). Its not an exaggeration to say that the entire system hinges on the values these two parameters take. A low θ_2 means that rarer bigrams will be chosen, while a low θ_1 value implies the final CER will be less.

4.2.1 Original Approach

Note that $S(\cdot)$ function has L_2 in it, so one can bound the value of S given θ_2. That is to say, if $L_2(w_{i-1}, w_i) \leq \theta_2$, then $\theta_1 \geq S(w_1 \dots w_i) \geq \alpha_1 L_1(w_i) + \alpha_2\theta_2 + \alpha_3\sigma_{\text{chr}}(w_1 \dots w_i)$, because α_2 is negative. Thus, for a given θ_1, the value of θ_2 can be bounded. The σ_{chr} and L_1 terms will be at least 0, then $\theta_1 \geq \alpha_2\theta_2$, or $0 \geq \theta_2 \geq \frac{\theta_1}{\alpha_2}$.

From the above discussion, it is possible to narrow down the range of values θ_1 and θ_2 takes for good passphrase generation. The range is not optimal, but works good enough in most cases. In the Mascara system, the value of θ_2 is set to 0.8 times the smallest value of L_2, thus effectively allowing a range of 20% leeway from the lowest bigram probability. With the help of the above relation and some manual tuning, a good θ_1 value was found to be 0.5.

The way the parameters were chosen is by first deciding on a list of the parameter pairs. Then for each pair, the system is used to generate passphrases and these passphrases are stored. Then, finally, all these set of passphrases undergo manual scrutiny from multiple human reviewers and the passphrases are evaluated subjectively as well as taking in the corresponding CER and Guessrank values into account.

It goes without saying that this approach is not efficient and accurate enough. Without even talking about the time and resources needed to manually evaluate each set of passphrases generated, just the bias introduced is enough to let the results go awry. This doesn't even consider the scalability of the approach. If more parameters were introduced, or a higher precision for the values of the parameters are needed, this method will obviously won't work. Thus, we need an alternate approach.

4.2.2 Grid Search

From the previous discussion, we can see that the problem arises from the fact that human intervention is required for the evaluation of passphrases, or more precisely, rank the systems with different parameters. To forego human evaluation, we need some form of fast assessment that can be automated. If that is possible, we can just carry out a grid search across a suitable range corresponding to the parameters.

Towards that direction, we introduce a new evaluation metric μ for each system. Higher the value of μ, the better the passphrases generated. μ can be considered as the average of the ratios of $\log_{10}$(GuessRank) and CER values for each passphrase with slight modification. Before we move on to the modification, we note that this metric incorporates all the required attributes an evaluation metric must have. It tries to encourage high guessrank and low CER values.

But, just the average alone is not perfect. Because, an anomalous passphrase can have a very low guessrank, but it can significantly contribute to the final μ value if it has low enough CER. This is not something we want to see. The same can apply for passphrases with very high CER. To nullify the contributions of such passphrases to the final result, we threshold the guessrank and CER values to their top and bottom 99%, respectively. The other 1% values of guessrank and CER will be made 0 and ∞, respectively. This will handle all the anomalous cases. Note that we don't have to worry about this happening with generic passphrases because of the constraints imposed (θ_1 and θ_2) while generating the passphrases.

Now things have become simple. The only thing left is to decide on a range for θ_1 and θ_2 on which to carry out the grid search. Since θ_2 imposes the restriction on the bigram probabilities, we can have the range of θ_2 between some percentage of the minimum value of L_2 across all bigrams. And θ_1's lower bound can be calculated with the relation discussed above, and upper bound can set to a fixed constant (this fixed constant is related to the general CER values of the dataset. Can be estimated for every dataset). After deciding the step for the search based on the precision requried, we can carry out the grid search and then rank the top k (can be 5 or 10, depending on the user) systems based on the μ values, whose passphrases can be manually evaluated and analysed. We notice that this approach has addressed all our previous concerns.

4.3 New Readable Passphrase Algorithm, MMAP

As we know, its very important to evaluate a system exhaustively in all directions to know all its perks and shortcomings. Future works can work in improving the existing disadvantages, if they have been put to light through a proper evaluation. In this area, although Mascara uses some of the traditional and popular approaches and algorithms as benchmarks, none of them are perfectly geared to target the generation of memorable passphrases. Towards that we try to introduce a popular tool from the internet, MMAP [1].

4.3.1 The Algorithm

This algorithm started off as a random passphrase generator, until it gained popularity for generating rememberable yet secure passphrases. The high memorability of the generated passphrases comes from the fact that this algorithm is geared towards generating passphrases that are linguistically sound. That is, the algorithm tries to generate meaningful sentences. Although it uses a lot of hard coded information which is difficult for machines to come up with, thus putting its versatility and scalability to question, it nevertheless does a very good job of producing meaningful (and thus memorable) passphrases.

The algorithm hinges primarily on two facts:-

- **Dictionary:** A vocabulary of around 20,000 words, that is manually acquired (hand picked), is fed to the algorithm. The collection consists of a wide variety of commonly used words, rarely used words, prepositions, pronouns, etc. (stopwords are not removed). But the most important feature of this dictionary is that it it segregated on the basis of its Parts of Speech (PoS) tags. Some of these classes are Nouns (proper, common), Verbs, Adverbs, Adjectives, etc.

- **Templates:** The algorithm is hardcoded with around 27 semantic templates that define how the various PoS come together. These templates were created from linguistic rules that establishes the proper syntactic structures for sentences in English. One such template would be $< Noun \mid Verb \mid Noun >$. But it is not as simple as that. There are various probabilities associated with each part of the template to choose the proper subclass and other adjustments that are made depending on situations. All these rules are manually incorporated into the algorithm.

  This should have given a brief idea of how the algorithm approaches the problem of generating meaningful passphrases. Basically, depending on the requirement on the strength of the passphrase, a suitable template is chosen among the defined templates, probabilistically. Then for each part of the template a suitable class and subclass is decided upon (again depending on the defined probabilities), and the words in the corresponding class are chosen randomly. Finally, the generated passphrases are fine tuned with the addition of suitable articles, conjunctions, etc. based on certain rules and thus we arrive at the final passphrase. The algorithm is simple as well as complex in its own way.

4.3.2 Guessrank and Memorability Estimation

Now that we are familiar with the algorithm, the next step that has to be carried out is to incorporate this passphrase generator into our pipeline for evaluation. That is, we need to decide upon whether we are going to stick with the same guessrank and CER measure that we have used previously or if something has to be changed.

Guessability. One solid way to measure guessability is to estimate the expected number of guesses (tries) an adversary has to carry out before he arrives at the actual passphrase (as we have presented in Chap 3.1.4). But the way we approach the estimation can be different. We have previously seen a Markov model of estimating the guessrank. It has worked for previous cases, and even in the case of MMAP, it works for passphrases of smaller length. But for larger passphrases, an adversary can leverage the templates and limited vocabulary in his favour and perform an exhaustive search for each template until he arrives at the passphrase. The order of the templates chosen can be random or priority based (on the number of passphrases). But, for this approach, one has to precalculate the number of passphrases that could be generated in each template, which is not exactly difficult. The reason why this approach is chosen because, it gives off lower guessranks for larger passphrases as compared to Markov estimation, and an adversary would obviously strive for guessranks to be as low as possible.

Memorability. There is no real motivation to change the approach we have previously taken to measure memorability for the other algorithms. For one, we still do not have any better way to quantify memeorability than CER at the moment. And second, CER, unlike guessrank, cannot be computed in different ways and be compared, as the CER has a proper method of calculation asssociated with it. Hence, we stick with the same CER that has been calculated with the help of unigrams and bigrams.

Now, we have seen all the modifications that has been made to the previous Mascara system. The next chapter evaluates the impact that these modifications have on the system.

Chapter 5

Evaluating Passphrases

Now that we have discussed the various changes that have been incorporated into the Mascara system, we will now see what impact it leads to. In the first section, we will discuss about the improvement on runtime and rejection rate before and after removing phrasemachine. In the second section, we will see MMAP and Mascara being compared along with all the previous benchmarks and make objective remarks about the results.

5.1 Impact of Phrase Machine

So, we have removed the phrase machine. Does it do something? Yes, it does. In fact, the measurements taken before and after showed a significant change. Since we were aiming to optimise the passphrase generation runtime, we will have a look at the figures before and after the removal of phrase machine.

Length	Runtime$_a$ (in s)	Rejection$_a$	Runtime$_b$ (in s)	Rejection$_b$
4	0.08	4.34	8.7	68.4
5	0.09	5.85	18.2	91.8
6	0.10	7.24	28.1	112.4
7	0.13	11.62	43.1	201.7

Figure 5.1: *Values of runtime and rejection count before and after removing PhraseMachine, for various lengths*

From the Figure 5.1, we can see the average run time and rejection count for passphrases of different lengths, before (X_b) and after (X_a) removing the *PhraseMachine*. The rejections indicate a count, whereas runtimes are represented in seconds. As we can see, there obviously is a decrease in both of these values after making the change. The rejections that occur after the removal are from the heuristics inserted to make up for the loss of *PhraseMachine*. In fact, as the length of the passphrase increases, the difference in these values increases. And it does exponentially. This is because, bigger the length of the passphrase, higher the probability that at least one word of it is responsible for the rejection. Thus, we can see that, there is a significant improvement in terms of running time.

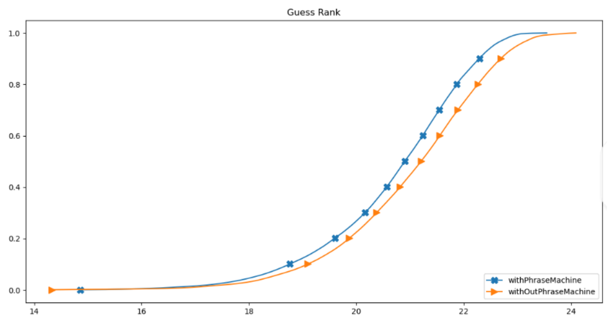

Figure 5.2: *CDF of guess rank (in log 10 scale) for passphrases of length 6 generated with PhraseMachine and without it.*

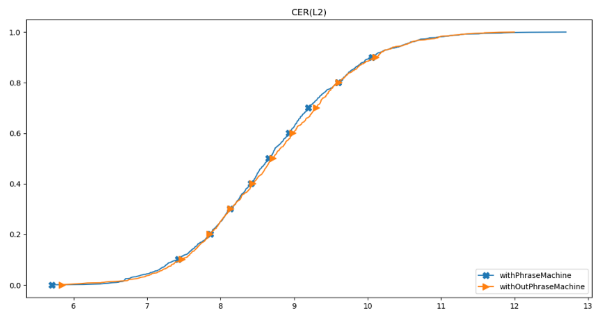

Figure 5.3: *CDF of CER for passphrases of length 6 generated with PhraseMachine and without it.*

But we have to consider the generated passphrases as well. We cannot accept the change, if the runtime optimsation occurs at the cost of Guessrank and CER. Thus, after running both instances of the system and comparing the generated passphrases of length 6, we found that the guessrank has even increased compared to when the *PhraseMachine* was included, while there is a slight decrease in CER. This is shown in Figure 5.5 and Figure 5.3. While considering both these aspects, we deem that after the change, the Mascara system has improved by a lot.

5.2 Evaluation with MMAP

The following section compares the performance of all benchmarks, which now includes MMAP as well, with Mascara.

5.2.1 The Benchmarks

The following set of passphrases were considered for evaluation where the guessrank and CER of the generated passphrases were calculated and plotted.

- **Diceware**: This generates random passphrases by choosing random words from a vocabulary. We choose the vocabulary of the Wiki-5 dataset (as was done during the Mascara evaluation), removing 5% of the most frequent words and least frequent words. The resultant vocabulary had 360,868 words.
- **UserPP**: Identified several passphrases used by users from prior password leaks. This is then considered as a separate set of passphrases.
- **Markov**: A bigram Markov model was trained over words from the Wiki-5 dataset. The passphrases generated by this model are what we called Markov. For estimating probability according to this model, backoff smoothing [43] is used, where we use the product of unigram probabilities of the words in the bigram, if the bigram is not present in the training data.
- **MMAP**: Set of phrases generated by MMAP, as described in Chap 4.3.
- **Mascara**: Passphrases generated by Mascara using the parameters optimised by grid search and removing the phrase machine.

Random samples of passphrases from each set mentioned above are shown in Figure 5.4.

5.2.2 Guessability

We show the (estimated) guess ranks of the passphrases in our test data in Figure 5.5. Diceware passphrases have the highest guess ranks - 50% of passwords will require at least 10^{16} guesses - as expected given that they are sampled uniformly from a large dictionary of 3.6×10^5 words. The UserPP and the Markov passphrases are the easiest to guess - 50% passwords can be guessed in less than 10^{10} guesses. Again, not surprising, given that the users often pick predictable phrases from English texts, and the attack algorithm is trained on Wiki-5 dataset as well. The similar guess rank distribution between Markov passwords and UserPP (we reject the null hypothesis using a two-sample KS test [44], with $p = 0.054$) also attests to that same fact.

Mascara provides a modest improvement, requiring more than 10^{12} guesses to guess 50% of passphrases, over UserPP and Markov, but significantly less than that of Diceware.

Type	Samples
Diceware	dreamscape manchuria dervish verbally whinging ferries portmore permanency downcast epilogue guarding richemont
UserPP	dont forget the password just another happy ending ride with the wind
Markov	the problem standard kit during ultraviolet signals beamed there improvements achieved worldwide
MMAP	how does its 1 shire milk a jack the frothy thing faxed the aphorism the rivalry was misspelling prior to our eyeballs
Mascara	many local fiscal rights lee sung before god jackal with prayer requests

Figure 5.4: *Three randomly sampled passphrases from each group of passphrases we consider for evaluation.*

Guessrank for MMAP is pretty high compared to the other variations bar Diceware, but it has its limitations when it comes to passphrases with higher length. We will discuss about this shortly.

5.2.3 Memorability

The distribution of CER of the passphrases are shown in Figure 5.6. As expected, Diceware passphrases have the highest CER meaning that users are likely to have a hard time remembering them and entering them correctly. It is hypothesized that the lack of any syntactic structure is responsible for such high CER. UserPP passphrases seem to perform the best and are expected to do so, given users, in general, choose highly common phrases, quotes, and song or movie titles.

CER values of the Mascara passphrases are between UserPP and Diceware. Mascara optimizes the passphrases to have lower CER values. We see that 50% of Mascara passphrases have CER less than 4.2 - (16% worse than the median CER of User-bg passphrases, and 27% better than the median CER of Diceware passphrases). MMAP passphrases have much better CER because of the syntactic structure the generated passphrases follow. This is supported even more by the fact that a lot of unigrams are popular as the dictionary is handpicked.

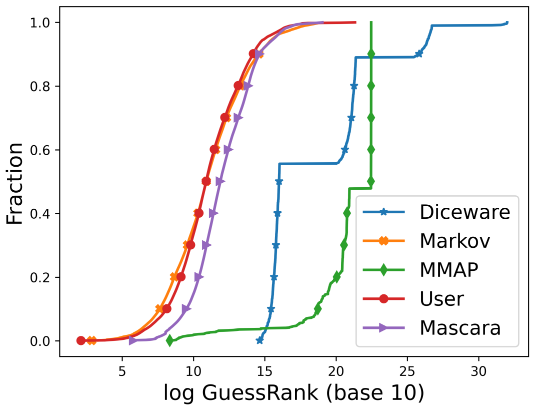

Figure 5.5: *CDF of guess rank (in log 10 scale) for passphrases.*

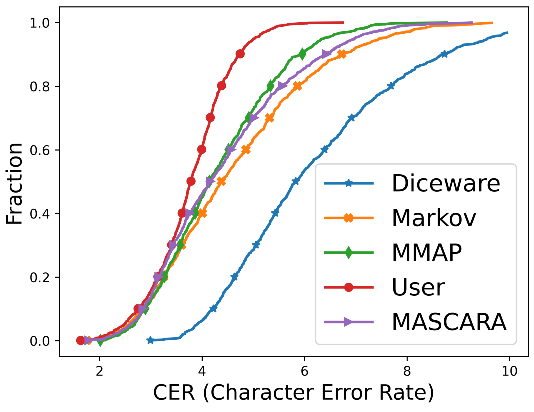

Figure 5.6: *CDF of CER(Character Error Rate) as rate of errors per character.*

So, it seems MMAP performs better in all areas and we can just directly conclude that MMAP is better than Mascara. But is it really so? The following section discusses this fact.

5.2.4 Mascara vs MMAP

Since, the main factor for involving MMAP into the evaluation system is to check how it compares with Mascara, it will be anti-climatic if we finish the discussion without mentioning it.

Now, considering the Guessrank factor, we can see that MMAP performs significantly better than Mascara. In fact, it seems to be even comparable to Diceware, which doesn't even take memeorability into account. If we include the CER as well, it just makes matters worse, as we know the underlying principle in MMAP is to make the passphrases easier

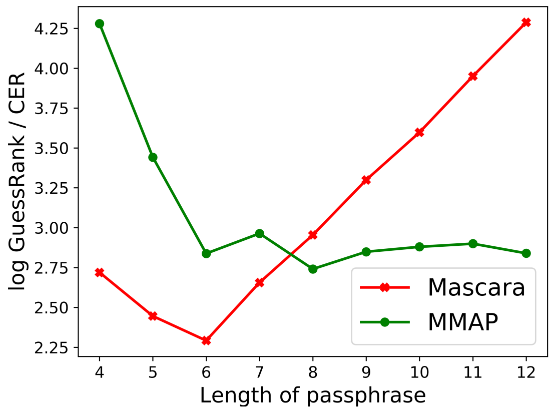

Figure 5.7: *Average ratio of log guessrank and CER of passphrases of different lengths generated by Mascara and MMAP.*

to remember and CER enforces this point.

So, the key point of the discussion relies on the length of the passphrase. So as to compare the generated passphrases with u, we have restricted the length of the passphrases from other systems to be very small as well (an average of 4). Such short passphrases alone do not convey much significant details. As one knows, increasing the length of the passphrase increases its Guessrank as well (a longer passphrase is harder to guess). This is common in pretty much all passphrase systems. But MMAP reaches a saturation in its Guessrank around word length of 6. This means that, there is no increase in security even if we change the passphrases from length 6 to length 10, whereas Mascara notes a significant improvement. This is mainly because of the fact that the MMAP system uses fixed templates, whose passphrases are finite and exhaustible. Thus an increase in length offers no obvious advantages to its passphrases.

On the perspective of CER, the difference between the values of CERs between Mascara and MMAP, significantly decreases. All in all, as we increase the length of the passphrase, Mascara gains significant advantage over MMAP. This is shown in Figure 5.7, which plots the metric used to rank the systems in grid search for Mascara and MMAP, for varying lengths. As we can see, beyond length 9, the passphrases generated by MMAP could not even be compared with Mascara.

Thus, the question of which is better between Mascara and MMAP lies on the requirement of the user, but in general, as security is deemed more important these days, users tilt towards passphrases of higher length, thus being biased towards Mascara.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work, we aim to improve upon the previously proposed Mascara system, which tries to address the problem of trying to automate the generation of memorable passphrases while also being secure. We have seen that how passphrases play an important role in today's world where security being compromised is not a rare incident. Passphrases are the de-facto approach for authentication in many places, and these passphrases could be used as a context for password generation. Thus, solving the problem of generating memorable (users can do so, but easily guessable) and also secure (machine generated passphrases, but compromises on memorability) passphrases addresses almost all of today's regular authentication mechanisms.

Mascara does just that. It proposes a framework for quantifying the subjective measure of security and memorability. With the above said frameworks as well as using a constrained Markov generation model, Mascara is able to generate secure passphrases that are very memorable. When evaluating the system and comparing it with a lot of traditional approaches, we see how the passphrases generated by Mascara maintains a better balance (that can be controlled) between memorability and guessability.

But, it is observed that the Mascara system can be significantly improved, both in terms of performance and evaluation. First, we focus on improving the runtime of the system, by removing the almost redundant linguistic check and adding certain heuristics, which speeds up the generation process by hundred times on average, without much compromise on memorability nor guessability. It goes without saying how significant this improvement is.

Secondly, we have noticed that Mascara chooses its parameters related to the constrained Markov generation manually. That is, the parameters were set and then the corresponding passphrases generated were compared subjectively to choose the best among them. This obviously is not feasible when finer precision on parameter values are needed, or if additional parameters are being introduced in the future. To solve this problem, we have developed evaluation metric for the generated passphrases, which can then be used to automate the search for optimal parameters by performing a grid search and ranking the parameters corresponding to the generated passphrases. Even, in the current scenario, after performing a grid search, we have observed significant improvements on the Mascara generated passphrases.

Finally, we felt that the evaluation of the passphrases generated by the system was not really exhaustive. To be precise, the other approaches used as benchmark, might be popular and conventional, but none really generate passphrases keeping memorability in mind, and hence do not serve as a good enough comparison for the memorability factor. Towards that extent, we introduce another popular approach MMAP [1], which also aims to secure passphrases, but keeping in mind the memorability factor as well. We show how to approach the estimation of guessrank in this system, and then move onto compare the performance of Mascara against this algorithm. We notice that although MMAP performs much better for passphrases of smaller lengths ($<=$ 5 words), when the length of the passphrases increase, the performance of MMAP drops when compared to Mascara.

6.2 Future Work

Even with all these enhancements, we feel that Mascara still has a lot of undug potential which could be utilised to improve the system even further. Some promising avenues that could be targeted are:-

- One of the most important part of the Mascara system, is the training corpus used. The currently existing dataset is reasonably good, especially with all the preprocessing. But, examining the processed corpus, we will find that the bigrams have a long tail. That is, the number of occurrences of certain bigram is only once, and such bigrams aren't exactly rare. It is obvious that such tails will hamper the resultant Markov model's generation process. We need to do something about the tail, or explore new datasets.

- We have removed the use of phrase-machine, which tests for linguistic correctness of the final generated passphrase and have replaced it with simple heuristics. We could introduce more powerful heuristics, that uses the intermediate passphrases, like guiding the incremental addition with PoS (parts of speech) tags. Since, we are handling intermediate passphrases instead of the final generated passphrase, the execution time will not be impacted by a lot.

- Another addition that could potentially improve the system a lot is involving trigrams. Involving trigrams not only will increase the accuracy of the estimation of guessability and memorability, we can have more control over the generation process. But, the way trigrams are incorporated into the system must be handled with care because of the significant overhead it might have on the generation process.
- This might have occurred to a lot of people, but Character Error Rate (CER) is just a measure on how much error in general it takes to type in the passphrase. This might sound reasonable on the surface, and could even be correlated to actual memorability, but it surely doesn't indicate a proper memorability. To that extent, we might need some other estimation of memorability that tries to actually measure how rememberable a passphrase is or something that is similar. This could even be an extension or improvement over the current CER.

References

[1] Readable passphrase. `https://makemeapassword.ligos.net/generate/readablepassphrase`.

[2] Simon S Woo and Jelena Mirkovic. Improving recall and security of passphrases through use of mnemonics. In *Proceedings of the 10th International Conference on Passwords (Passwords)*, 2016.

[3] Shayan Eskandari, Jeremy Clark, David Barrera, and Elizabeth Stobert. A first look at the usability of bitcoin key management. *arXiv preprint arXiv:1802.04351*, 2018.

[4] Alexa Huth, Michael Orlando, and Linda Pesante. Password security, protection, and management. *United States Computer Emergency Readiness Team*, 2012.

[5] ssh-keygen(1) - linux man page. `https://linux.die.net/man/1/ssh-keygen`.

[6] AG Reinhold. The diceware passphrase home page (1995). `https://theworld.com/~reinhold/diceware.html`.

[7] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the eighth symposium on usable privacy and security*, pages 1–20, 2012.

[8] Joseph Bonneau and Ekaterina Shutova. Linguistic Properties of Multi-word Passphrases. In Jim Blyth, Sven Dietrich, and L. Jean Camp, editors, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 1–12. Springer, 2012.

[9] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authen-

tication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE, 2012.

[10] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26, 2014.

[11] Marjan Ghazvininejad and Kevin Knight. How to memorize a random 60-bit string. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015.

[12] Cynthia Kuo, Sasha Romanosky, and Lorrie Faith Cranor. Human selection of mnemonic phrase-based passwords. In *Proceedings of the second symposium on Usable privacy and security*, pages 67–78, 2006.

[13] Paul A Grassi, Michael E Garcia, and James L Fenton. Draft nist special publication 800-63-3 digital identity guidelines. *National Institute of Standards and Technology, Los Altos, CA*, 2017.

[14] Elizabeth Stobert and Robert Biddle. The password life cycle: user behaviour in managing passwords. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 243–255, 2014.

[15] Simon S Woo. How do we create a fantabulous password? In *Proceedings of The Web Conference 2020*, pages 1491–1501, 2020.

[16] Mahdi Nasrullah Al-Ameen, Matthew Wright, and Shannon Scielzo. Towards making random passwords memorable: Leveraging users' cognitive ability through multiple cues. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2315–2324, 2015.

[17] Marco Antônio Carnut and Evandro Curvelo Hora. Improving the diceware memorable passphrase generation system. In *Proceedings of the 7th International Symposium on System and Information Security. São José dos Campos: CTA/ITA/IEC*, 2005.

[18] Nikola K Blanchard, Clément Malaingre, and Ted Selker. Improving security and usability of passphrases with guided word choice. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 723–732, 2018.

[19] Daniel V Klein. Foiling the cracker: A survey of, and improvements to, password security. In *Proceedings of the 2nd USENIX Security Workshop*, pages 5–14, 1990.

[20] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1242–1254, 2016.

[21] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 417–434. IEEE, 2019.

[22] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. Design and Evaluation of a Data-Driven Password Meter. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17. Association for Computing Machinery, 2017.

[23] Steven Van Acker, Daniel Hausknecht, Wouter Joosen, and Andrei Sabelfeld. Password Meters and Generators on the Web: From Large-Scale Empirical Study to Getting It Right. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, March 2015.

[24] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 175–191, 2016.

[25] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How does your password measure up? the effect of strength meters on password creation. In *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, 2012.

[26] Kok-Wah Lee and Hong-Tat Ewe. Passphrase with semantic noises and a proof on its higher information rate. In *2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)*, pages 652–655. IEEE, 2007.

[27] Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1:*

Contributions to the Theory of Statistics. The Regents of the University of California, 1961.

[28] Xianyi Gao, Yulong Yang, Can Liu, Christos Mitropoulos, Janne Lindqvist, and Antti Oulasvirta. Forgetting of passwords: ecological theory and data. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 221–238, 2018.

[29] Yimin Guo, Zhenfeng Zhang, and Yajun Guo. Optiwords: A new password policy for creating memorable and strong passwords. *Computers & Security*, 85:423–435, 2019.

[30] Joseph Bonneau and Stuart Schechter. Towards reliable storage of 56-bit secrets in human memory. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 607–623, 2014.

[31] Rahul Chatterjee, Joanne Woodage, Yuval Pnueli, Anusha Chowdhury, and Thomas Ristenpart. The typtop system: Personalized typo-tolerant password checking. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 329–346, 2017.

[32] Mark Keith, Benjamin Shao, and Paul John Steinbart. The usability of passphrases for authentication: An empirical field study. *International journal of human-computer studies*, 65(1):17–28, 2007.

[33] Cristian Danescu-Niculescu-Mizil, Justin Cheng, Jon Kleinberg, and Lillian Lee. You had me at hello: How phrasing affects memorability. In *Proceedings of the ACL*, 2012.

[34] I Scott MacKenzie and R William Soukoreff. A character-level error analysis technique for evaluating text entry methods. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 243–246, 2002.

[35] Per Ola Kristensson and Keith Vertanen. Performance comparisons of phrase sets and presentation styles for text entry evaluations. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pages 29–32, 2012.

[36] William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 175–191, 2016.

[37] Wikimedia dump. `https://dumps.wikimedia.org/enwiki/latest/`.

[38] Rahul Chatterjee, Anish Athayle, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. password typos and how to correct them securely. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 799–818. IEEE, 2016.

[39] Luis A Leiva and Germán Sanchis-Trilles. Representatively memorable: sampling the right phrase set to get the text entry experiment right. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1709–1712, 2014.

[40] Joseph Bonneau. *Guessing human-chosen secrets*. PhD dissertation, University of Cambridge, 2012.

[41] Matteo Dell'Amico and Maurizio Filippone. Monte carlo strength evaluation: Fast and reliable password checking. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 158–169, 2015.

[42] Abram Handler, Matthew Denny, Hanna Wallach, and Brendan O'Connor. Bag of what? simple noun phrase extraction for text analysis. In *Proceedings of the First Workshop on NLP and Computational Social Science*, pages 114–124, Austin, Texas, November 2016. Association for Computational Linguistics.

[43] Back-off model. `https://en.wikipedia.org/wiki/Katz's_back-off_model`.

[44] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.