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
April, 2022

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
Assistant Professor
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur,
West Bengal

April, 2022
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 at the present. However, prior studies have established that it is inherently difficult for users to generate secure passwords. To that end, passphrases (human-readable phrases consisting of a sequence of words) are often considered a potentially useful alternative to passwords, where memorable yet secure authentication secrets can be generated. In fact, passphrases are the only go-to in many cases where security is deemed extremely important and moreover, passphrases can be used as a context to generate more secure passwords.

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, as we show in this work, machine-generated passphrases are difficult to remember despite being secure. To that end, Mukherjee [1] proposed MASCARA. It is a systematic approach for the generation of secure yet memorable passphrases which leverages a memorability-guessability framework developed to find an optimal tradeoff between security and memorability.

However, the work has some huge drawbacks. The current guessability framework in use guarantees only an upper bound in security far off from the actual measure, whereas the memorability system has been developed purely from theoretical claims. Furthermore, the performance of the generation of passphrases from MASCARA in terms of the runtime is extremely lackluster, while the tuning of the model in itself completely requires manual intervention, resulting in a lack of scalability and potential bias in the system.

The work in this thesis aims to improve upon all these issues starting with the almost complete overhaul of the memorability-guessability measurement framework, then proposing various modifications to the core of MASCARA like removing redundant heuristics and improving the generation complexity, all of which results in a drastic improvement in its performance. Towards the end, we evaluate the passphrases generated by the enhanced MASCARA using the new framework for evaluation, which highlights the improvement in the security of the MASCARA generated passphrases over in-use user-generated passphrases, all the while mitigating shortcomings found within in-use system-generated passphrases. The longitudinal user study carried out demonstrates that users have the highest memory recall and lowest error rate while authenticating with MASCARA passphrases when compared to other system passphrases.

Contents

Chapter 1

Introduction

Billions of Internet users authenticate themselves across the Internet using login credentials or authentication secrets. Passwords are one of the popular types of authentication secrets. However, password creation and management are a big problem even these days despite their popularity. Users are bad at picking passwords that are strong and hard to guess but also easy to remember and enter. Moreover, passwords are not without problems as shown in a lot of prior work and evidenced by the huge corpus of password data available on the internet; this same corpus enables targeted guessing attacks which can guess over 32-73% of passwords within 100 attempts [2]. To that end, *Passphrases* can be considered an alternative approach to generate memorable yet strong authentication secrets. Passphrases are just a group of words mashed together to form a phrase (e.g., "correct horse battery staple") as opposed to passwords which are just a sequence of characters.

In particularly sensitive cases where security is extremely important like the master secret for password managers [3], to secure their Cryptocurrency wallets (such as brainwallet [4]) or even protecting ssh keys [5], passphrases are the primary choice of authentication. In a lot of cases, a password could be created with the passphrase as a base (e.g., a mnemonic created from a passphrase can be used as a memorable password [6]), thus making it a lot more memorable.

In this work, we have observed that there are two ways of generating passphrases in the wild—user-generated and system-generated. User-generated passphrases are memorable, often because of their sentence/grammatical structure being aligned with natural language [7, 8, 9, 10, 11]. However, they are also relatively easily guessable due to this very structure [12, 13], thereby increasing their chance of being attacked. Specifically, some of the prior works [14, 12] have demonstrated that passphrases manually chosen by users according to their preferences are very memorable but weak in terms of security, which an

attacker can leverage. To that end, system-generated passphrases are proposed—their security advantages include resistance to targeted impersonation, throttled and unthrottled guessing and leaks from other verifiers [15]. Unfortunately, the use of system-assigned passphrases comes at a cost to memorability [14, 16].

Specifically, one of the current systems that generate passphrases for practical use is Diceware [17, 14], which involves choosing random words from a given wordlist. However, Shay et al. [14] showed that Diceware passphrases are error-prone to use—memorability of Diceware passphrases is similar to randomly generated passwords.

During our exploration, we found another improved variation of diceware, *Template based diceware*, which is used in the wild [18]. This variation tries to resolve the problem of memorability of passphrases generated from Diceware using a small number of manually curated syntax templates and a wordlist. However, our analysis revealed that even this approach suffers two major drawbacks—(i) it imposes a finite bound on the passphrase strength due to the usage of a handful of specific syntax templates and (ii) need for non-systematic manual effort to extend passphrase generation with more templates (Section 4.3), necessitating a thorough re-design of the approach. Several works took an alternative approach—they tried to use various learning techniques like implicitly learning post-passphrase generation to improve their memorability [16]. Complementary to those efforts of enhancing memorability by user-learning, we ask: *Is it possible to develop a simple automated approach for producing system-generated passphrases of arbitrary length, which is memorable by abiding grammar/sentence structure, yet hard for an adversary to guess and address the shortcomings of existing passphrase generation systems?*

In this work, we present significant contributions that answer this question positively. For this, we leverage a previously established work by Mukherjee [1], MASCARA, which also tries to address the problem of memorable yet secure passphrases through systematic generation with the help of a guessability-memorability measurement framework. But this system is incomplete in terms of the established measurement framework, as well as hampered by various performance issues. We try to address the above question by resolving the various problems surrounding the system and presenting a significantly improved MASCARA for the generation of secure yet memorable passphrases.

Note that, in line with the previous works (and for a better comparison with state of the art), our exploration considered analyzing and generating English passphrases [16, 12, 19]. We leave extending the system to other languages as future work.

1.1 Contributions

The key contributions of this work are:

- We have revamped the guessability measurement framework and practically verified the system of memorability proposed by Mukherjee [1]. The original guessrank framework only estimates an upper bound on the security of the generated passphrases, which was proved to be almost ineffective against the current real-world approaches employed by professional crackers [20]. We have modified and extrapolated the idea proposed in the earlier work for estimating the security of passwords in real-world scenarios as the guessability framework for the passphrases. In the case of memorability, the various parameters used as a factor in the calculation in CER has only a theoretical claim to back them up in the proposed work. We have carried out experiments to verify those claims by establishing the correlation between those parameters and CER, and also provided enough literature references for the same.
- We analyze the different classes of passphrases in the wild currently in use. We leverage the passphrase dataset retrieved from prior password leaks by Mukherjee [1], and examine them. We also carry out a deep survey of popular password managers, one of the platforms where passphrases are widely used, and identified two of the most common and popular systems that are employed for passphrase generation—Diceware and TemplateDice. These systems are then dissected and their various properties and limitations are then analyzed and presented.
- As the core of the work, we propose various enhancements to the algorithm of passphrase generation used in MASCARA. We first remove the redundant and time-consuming phase of passphrase screening that uses *PhraseMachine*, which rejects passphrases until a meaningful one is produced. Although this is done to enhance memorability, the tradeoff between performance and the quantitative increase in memorability offered is not equivalent. This alone has offered a 25 times speedup of the previous system. Furthermore, we improve the actual generation algorithm by completely changing the linear complexity for every passphrase to logarithmic complexity for every passphrase with a one-time linear preprocessing. The said complexities are over the vocabulary size used for the generation and are of the order of 10^5. This has resulted in a humongous improvement in the performance of over 1000 times over the system after the removal of *PhraseMachine*.
- We have provided a systematic approach for tuning the system parameters which govern the guessability and memorability of passphrase generation. Previously in MASCARA, these parameters were manually selected after observing the resultant passphrases gen-

erated across a wide number of choices. But, this approach falls through if the required resolution for the parameters is increased, or if the system is expanded to accommodate more parameters. Keeping that in mind, we have introduced a new measure μ which incorporates both guessrank and CER, and can be used to automatically rank a system for a set of parameters based on the μ value of the generated passphrase. If needed, manual evaluation can be carried out for the top 5 or 10 parameter choices as required.

- We evaluate the passphrases generated by the enhanced MASCARA, including the previous baselines, as well as introduce the new TemplateDice, using the new guessability and memorability framework. With this, we have established the advantages the system offers in terms of making a proper tradeoff between guessability and memorability as compared to other classes of passphrases that focus particularly on one of these aspects. We have also shown how the system overcomes the various limitations of TemplateDice. A user study in the form of a survey was carried out to test how users remember passphrases, which further corroborates the experimental results.

The rest of the work is organized as follows: Chapter 2 provides background and the necessary knowledge on passphrases, as well as discusses some prior works in this avenue. We move on to re-establish a proper measurement framework for the guessability and memorability aspects of passphrases in Chapter 3. Then we explore the current passphrases in use, both user-chosen and system generated, and their properties and limitations in Chapter 4. In Chapter 5, we introduce the basic system design of the previously established MASCARA. Then, in Chapter 6, we move to the core of this work, where the various modifications and enhancements to the system and the evaluation pipelines are presented. The results for the enhancements as well as the evaluation of the passphrases using the new guessability-memorability framework are discussed in Chapter 7. We finally conclude this work and provide other potential areas of improvement of the system in Chapter 8.

Chapter 2

Background and Related Works

In this chapter, we provide some essential background knowledge and information on some of the important works carried out till now in the field of system-generated passwords and passphrases. Along with this, we will also present our threat model and lay down important foundations for measuring memorability and estimating security for authentication mechanisms, with a particular emphasis on passphrases.

2.1 System-generated Passphrases

Many earlier works can be found that focus on passwords as the primary authentication mechanism for some of the critical as well as non-critical infrastructures [15]. Even now, passwords are commonly used on various platforms. Unfortunately, users have a tendency to choose passwords that are very predictable and use them across multiple accounts and platforms [21, 22]. As a result, the security and usability offered by passwords do not meet the current standards for important scenarios. To that end, a lot of proposals aiming to improve the security of such authentication mechanisms can be found, and they take approaches like designing better password meters or creating mnemonic passwords [23].

To address that, over the past years, alternative mechanisms have been suggested and passphrases are one of the most popular as they can also act as a complementary mechanism to passwords. A definition of a passphrase can be found in NIST as *memorized secret consisting of a sequence of words or other text that a claimant uses to authenticate their identity* [24]. In general terms, we can imagine passphrases as a combination of words pieced together to form a sentence, irrespective of whether it is syntactically or linguistically valid. Intuitively, owing to their closeness to natural language, a passphrase is much better than a password in terms of their memorability, while still having the potential to be extremely secure because of the large sample domain. A prior work [14]

has shown, with the help of a user study, that even a simple passphrase of three to four words randomly selected from a curated dictionary can be close to passwords generated using sophisticated methods in terms of entropy, which translates to security. Coupled of the fact that passphrases are much more memorable than passwords, the use case of passphrases is highlighted. In the current scenario, passphrases, especially system-generated ones, are quite common and are used in password managers, crypto-currency wallets [25], and ssh-keygen [5], demonstrating their versatility and popularity among the different applications requiring high levels of security.

We point out that some of the prior works on password creation often focus on generating *passwords* that are strong as well as easy to remember over the years. Some of the common techniques used to achieve the results are contextual cues, portmanteau, and mnemonic-based generation [26, 27, 23, 28]. These approaches have a common trait of associating some form of context (like a memorable phrase) to the generated passwords so that they are easy to remember. Isn't this the case with using passphrases to generate passwords? For all practical purposes, yes. Consequently, our work on improving the passphrase generation by MASCARA is very much in-line with the goal of generating memorable passwords, as we can always use such a passphrase in the above-said context for the password generation.

However, there has been no extensive study of systematically analyzing the guessability-memorability trade-off using a framework, which has resulted in a situation of not being able to properly ensure both aspects in the generated passphrase. The guessability framework defined by Mukherjee [1] in the previous work is not solid enough to withstand modern cracking techniques, while the memorability system does not have enough justification for the parameters that have been employed in it. In this work, we rectify the above issues by drastically revamping the guessability-memorability framework with the help of some of the previous works. Our approach is complementary to the works on passphrase learning approaches which aim to help users learn and memorize system-generated passphrases [29]. We create a simple framework that can generate secure passphrases with improved memorability compared to the state-of-the-art generation methods. The memory-enhancing techniques can be easily used with our method in a real deployment.

2.2 Security and Threat Model

Only when we can estimate the security of an authentication mechanism, we will be able to offer justification for its usage. Thus, measuring the security of some of the most common

authentication secrets is a well-studied topic [30]. Earlier works considered different approaches the attacker could employ and thus used Markov models, probabilistic CFG, neural networks, etc. for their guessability estimations [2, 31, 32, 33, 34, 35]. However, these techniques do not scale straightforwardly for passphrases as various considerations have to be taken into account depending on the type of estimation technique used and thus there is relatively less work in the domain of passphrases.

2.2.1 Using guessrank to measure passphrase security

Previous works have found numerous ways to increase the resistance of a passphrase to external attacks by raising its entropy via either introducing semantic noises or by increasing the wordlist size [36]. However, once users are allowed to intervene in the selection of passphrases, they generally tend to opt for popular phrases commonly used, hampering the security of passphrases by a lot [23]. In all of the mentioned works, the security of passphrases is primarily measured by using entropy or surveys from users [37].

But entropy is a good enough measure only as long as there is no human involvement or bias in the generation of the passphrases. This is demonstrated by the later works and researchers where it is shown that *guessrank* as a metric to measure the security of passwords is much more suitable than entropy to get an idea of the security of a password [20, 38]. We can understand the guessrank of a password as the number of guesses or tries an adversary will make before they correctly identify the password. So, the higher the guessrank, the lower the guessability. Using these results and other earlier works on password meters [32, 33, 34, 35] as a foundation, we extrapolate the guessrank metric for quantifying how secure is a passphrase in our setup.

2.2.2 Our adversary model with *min-auto* approach

The notion of security for any authentication mechanism is only meaningful as long as we define the threat model being employed, as otherwise, we do not know how powerful is the adversary. In this work, we consider a powerful offline generalized untargeted adversary, similar to the threat model proposed by Mukherjee [1]. In the threat model, we are going with the fact that the adversary is privy to the core knowledge of the system like its primary training corpus and even the exact algorithm through which the passphrases are generated. However, the adversary does not know the randomized components employed by the system and they also do not target any particular user or group of users. One aspect of the threat model which is newly introduced in the work is the assumption that the attacker also has access to a huge dataset (e.g., over 10M) of passphrases generated

by the systems involved in the passphrase generation. This is reasonable considering the factors and parameters that the adversary has access to.

The primary challenge for the attacker is to generate an ordered list of possible passphrase guesses $\omega_1, \omega_2, ..., \omega_n$ that enable them to identify the target passphrase ω as early as possible (with the least number of guesses). The higher the guessrank of a passphrase the stronger the passphrase is (lower guessability). Given that there are multiple algorithms an attacker can use to guess a passphrase, we take a *min-auto* approach described by Ur et al. [20].

Although the *min-auto* approach is targeted toward passwords, it can be easily extended to any authentication mechanism. First, we used multiple passphrases cracking algorithms (n-gram word and character models trained over a large corpus of passphrases generated by the system under consideration), parameterized by a set of training data [39, 40]. For each algorithm, the guessrank will be the number of incorrect guesses the particular algorithm used to arrive at the correct passphrase. But, since the average guessrank for even a passphrase of medium strength is of the order of 10^{15} (Section 7.2.3), actually running the algorithms to find the guessrank is infeasible and thus we simulate the running of the algorithm using Monte Carlo estimations for our purpose (Section 3.2). Finally, following the previous work by Ur et al. we simply take the minimum of all guess ranks to arrive at our estimated guessrank. Ur et al. demonstrated that taking the minimum guessrank of all the automated approaches (called *min auto*) is a reasonable approximation of the real world cracking scenarios [20].

2.3 Measuring memorability of passphrases

Memorability is one important aspect that has to be considered in any authentication mechanism. Earlier works often tried to employ various parameters indirectly related to the behavioral patterns of humans to analyze how memorable are the passphrases. These parameters range from the frequency of passwords used, the time taken to log in, to even the keyboard pattern [41, 42]. Other works tried estimating memorability using methods like encoding random n-bit strings with the aim of creating ideal passwords or using chunks [22, 43]. All of the above-mentioned works tried to measure the memorability of passwords by employing the help of users in the form of user surveys [44, 7, 27, 14] instead of automating the process with a linguistic metric. These methods are not always reliable as they are dependent on the results of the user survey.

Moreover, most of these approaches trying to measure the memorability of passwords in a quantitative manner are not easily extendable to that of passphrases. Passphrases are

often characterized by their potential linguistic properties (the structure of the phrase) which are usually not found in passwords. Thus, we start with some previous work that focuses on the memorability of English phrases like the work by Danescu et. al. [45] trying to estimate the memorability of popular quotes from movies by performing a lexical analysis. Some Human-Computer interaction studies identified Character Error Rate (CER) as a good proxy for quantifying memorability of passphrases [46, 47].

Previously, Mukherjee [1] developed a memorability framework using these works as a foundation, identifying underlying factors that affect memorability. Using this framework, they optimize the generated passphrases using their algorithm to make the passphrases memorable (Section 5.2.1). But, it fails to justify the usage of the various parameters used to calculate the CER properly. To that end, we carry out experiments and employ the previous works involved in the memorability of English phrases to improve the memorability framework. Our longitudinal user study results demonstrated that users indeed memorize passphrases which is in accordance with the memorability model we proposed.

With this background, in the next chapter, we present our modified guessrank framework as well as the various justifications to validate the memorability model.

Chapter 3

Guessrank and Memorability

When it comes to authentication mechanisms, especially those that employ passwords and passphrases, two of their primary defining parameters are their security and usability. These two parameters will often be used to determine whether the password or passphrase is suitable for use, depending on the various scenarios. To that extent, we introduce the system of guessability and memorability. As the name implies, guessability is used to determine how secure a passphrase is, whereas memorability establishes a metric that helps us understand how user-friendly is a passphrase by estimating its difficulty to remember. Previously, Mukherjee [1] has also employed the use of a novel guessability and memorability framework, using which secure and memorable passphrases were generated. Although their framework for these parameters offers the basic functionality, the system of guessability does not go well with the current modern-world scenario where professional crackers have sophisticated means that can compromise the security of a passphrase. Moreover, the ideas used in memorability are not well reasoned out and justified. With the help of the details introduced previously, in this chapter, we will rectify the above issues by revamping the guessrank model and analyzing the memorability system in depth.

3.1 Memorability

One of the primary aspects we must take into consideration in system-generated passphrases is to make the generated passphrases easier for users to remember, i.e, increasing memorability. But to do so, we first must be able to quantify memorability, and this is where the memorability model developed comes into play. Here, we use the notion of character error rate (CER), which is the rate of error per character while typing in the text. Prior works [48, 46, 47] noted that CER is widely accepted as a proxy for memorability. An important

aspect of CER one has to understand is that it doesn't calculate the error when actually typing the passphrase, but the ratio of errors present in the final passphrases inputted by the user compared with the actual passphrase to the number of characters in the actual passphrase. For simpler understanding and execution, we use the edit distance for this purpose.

We consider CER as a good proxy of memorability since it can measure the accuracy (or quality) of a transcribed sentence. The more memorable a sentence is, the lesser the possibility of errors occurring in the final transcribed version. So, we focus on modeling memorability based on CER to make our generated passphrases easy to memorize.

To understand the factors that affect CER, we use the method adopted for text entry experiments by Leiva et. al. [48]. They select memorable phrases and sentences from a large corpus of text. They however tried to obtain memorable sentences, representative and exhaustive of the corpus, whereas we want to include those that might not be complete and representative of all passphrases users can memorize. Thus, the signals for memorability we want can be completely different. To decide the parameters of a phrase (passphrase in our case) that has to be considered for memorability, we experiment.

We used a dataset of 2,230 sentences, each of which has been annotated with the character error rate (CER) determined from a user survey [47]. We calculated various parameters for each phrase in the dataset like frequency of occurrence of words in the vocabulary, out of vocabulary words, the average frequency of occurrence, standard deviation of the word lengths of the phrase, number of words, etc. With these data, we find the statistically significant correlation of each feature concerning CER (using pearson's product moment correlation coefficient [49]) and found three promising signals, which are listed below.

- **Unigram probability (L_1) of a phrase.** Calculated as the sum of the log of unigram probabilities of occurrences of the individual words in the phrase. Phrases with a higher L_1 are likely to be more common, consisting of frequent words in the vocabulary, and hence are considered easier to memorize [50]. This can be seen from the fact that L_1 has $p \approx 0$ (very high statistical significance) and $r = -0.83$ (very high negative correlation) with respect to CER.

- **Bigram probability (L_2) of consecutive words.** Similar to L_1, L_2 is calculated as the sum of the log of bigram probabilities of all the consecutive pairs of words in the phrase. Phrases with a higher L_2 are more likely to be closer to the natural language and thus it will be easier for a user to remember, which is again supported

by its high statistical significance ($p \approx 0$) and high negative correlation ($r = -0.84$) with CER.

- **Standard deviation (σ_{chr}) of the word lengths .** If the words in the passphrase comprise of significantly different lengths, it leads to higher processing effort and cognitive load. This is corroborated by the fact that σ_{chr} has a very high statistical significance ($p = 10^{-24}$) and positively correlated concerning CER ($r = 0.25$).

Leiva et. al. [48] only considered unigrams to distinguish memorable sentences. However, in our work, we also used bigrams as identified by our experiment. Higher bigram probability also helps maintain syntactic structure, or the "lexical distinctiveness" of a phrase to increase memorability [45].

Note that for the computation of L_1 and L_2, we require a universal corpus. In this work, we use the Wiki-5 dataset (Section 5.1) developed and processed previously by Mukherjee [1]. The model was then fitted to the data using generalized linear regression with the above mentioned features. This yielded a good fit ($R^2 = 0.70$) for the CER estimate and we computed it for a passphrase s as $\text{CER}(s) = c_1 \cdot L_1(s) + c_2 \cdot L_2(s) + c_3 \cdot \sigma_{\text{chr}}(s)$, with $c_1 = -3.42\text{e}{-2}$, $c_2 = -6.46\text{e}{-3}$ and $c_3 = 1.19\text{e}{-4}$.

3.2 Guessability

After tackling the problem of memorability, we now handle the issue of guessability. As discussed in Section 2.2, the best way to measure the guessability (i.e., security) of a passphrase is to calculate its guessrank, which is, in fact, the estimated number of tries for an adversary to arrive at the correct passphrase. Recall that the higher the guessrank, the lower the guessability. The guessrank for a passphrase is calculated by using any cracking algorithm with the support of suitable training data [39, 40]. The algorithm is used to generate passphrases until it arrives at the correct passphrase, and the number of incorrect guesses it takes is the guessrank for that passphrase according to that model.

But, we note that, for passphrases with an even moderate level of security, their guessranks tend to be of the order of 10^{15}. The actual running of the algorithm over so many passes is physically and logistically infeasible just to calculate the guessrank of one passphrase. To that extent, we will be simulating the running of the cracking algorithm instead of actually running the algorithm. For most cracking algorithms, it is easy to associate a probability of generation for each passphrase according to the model used. We then leverage this probability of generation and employ a Monte-Carlo simulation [51] that uses this probability to estimate the guessrank of a passphrase without actually

running the cracking algorithm, thereby saving a significant amount of time and resources. The simulation is discussed below in brief.

In the pre-processing step, we generate n passphrases $\{\beta_1, \ldots, \beta_n\}$ from the target model $\mathcal{M}$ along with their estimated probabilities of generation. $\mathcal{M}$ is basically the model used in the cracking algorithm. We rearrange the probabilities in an array, sorted in descending order as, $A = [\mathcal{M}(\beta_1)....\mathcal{M}(\beta_n)]$, and create the rank array C, where it's i-th element value is computed as:

$$C[i] = \begin{cases} \lceil \frac{1}{nA[i]} \rceil, & \text{if } i = 0 \\ \lceil \frac{1}{n} \sum_{j=1}^{i} \frac{1}{A[j]} \rceil = C[i-1] + \lceil \frac{1}{nA[i]} \rceil, & \text{otherwise} \end{cases} \tag{3.1}$$

Here, the probability $A[i]$ corresponds to a rank $C[i]$, and for the simulation, we can extrapolate this property to all probabilities. Hence, to estimate the guessrank of a passphrase α, we find the largest j for such that $A[j] > \mathcal{M}(\alpha)$ through binary search. The guessrank is then calculated by taking a weighted average of the values of $C[j]$ and $C[j+1]$ (can be manually handled if $j = n$). Obviously, as n increases, the accuracy of the estimated guessrank also increases. For a more in-depth analysis of the algorithm, refer to the work presenting the Monte Carlo simulation [51].

The above process allows us to calculate the guessrank from any automated cracking algorithm that can generate passphrases along with their probability of generation, which includes most of the current state-of-the-art cracking algorithms [51]. With this one might realize that different guessranks can be obtained for the same passphrase, depending on the cracking algorithm used. This is where the earlier framework presented by Mukherjee [1] starts to fail. Previously, a single standard cracking algorithm is used to measure the security (i.e. guessrank) of passphrases generated by different systems and different algorithms. Thus the calculated guessrank just establishes an upper bound and does not talk about the lower bound, which could be orders of magnitude lower.

Previous research has also shown that a professional attacker using a semi-automated cracking process on a huge corpus of passwords can adapt to the dataset and thus is much more efficient than any known fully automated cracking algorithms [20]. This idea can be extrapolated to the domain of passphrases and thus using any single model for estimating the guessrank will overestimate the number of guesses to arrive at the correct passphrases as compared to the number of guesses required by a professional adversary in practice. We rectify this issue in this work, as presented below.

To resolve this issue, we use the concept of *min-auto*, which has been briefly discussed in Section 2.2. Here, we take into consideration multiple models generated from

different cracking algorithms which can be used to estimate the guessrank of passphrases and have been trained in suitable training data. Ur et al. [20] have shown that for each password taking the minimum of all guessranks estimated by the various models is a reasonable approximation to the actual guessrank needed in practice for passwords. Similarly, extending the idea to passphrases, we take the minimum guessrank of $n-$gram word and character models ($2, 3-$gram for words and $4, 5, 6-$gram for character) trained over a huge corpus of passphrases from the corresponding system to be evaluated, which is assumed to be available to the attacked (Section 2.2). Some system-specific models have also been considered to obtain a more accurate approximation of the guessrank, which has been explained in Section 6.3.2 and Section 7.2.3. With this, we finally have the guessrank used to determine how secure is a passphrase, irrespective of whether it is user-defined or system generated.

With the framework to determine the quality of passphrases properly established, we now move on to examine the different types of passphrases currently in use, as well as the motivation to work on MASCARA to optimize it.

Chapter 4

Passphrases in the Wild

In this chapter, we will examine the currently in-use passphrases and their properties to highlight the motivation for the work being carried out. Today, passphrase generation in the wild falls into two classes—user-generated and system-generated. User-generated passphrases (or *User* passphrases) are hand-picked by users and are potentially easier to remember for them, as that's their basic requirement for choice. System-generated passphrases are algorithm-generated passphrases (often parameterized by a training corpus or wordlist).

4.1 User Passphrases

To examine the class of user passphrases, we need to first acquire a dataset comprising of passphrases selected by users. Unfortunately, unlike passwords, where data breaches are not uncommon and a lot of which have surfaced publicly [52], there is no public dataset of passphrases available. So, Mukherjee [1] leveraged a simple idea in his work: password leak databases often contain long passwords that could potentially be passphrases without a proper delimiter. For this, they devised a segmentation algorithm to identify passphrases from password leak datasets and construct the first user-chosen, in-use passphrase database.

We now describe the process with which the author carried out the extraction of the passphrases using the password leak database. They started with a bundle of prior data breaches that surfaced in 2018 hosted by 4iQ security firm [52, 31, 53]. The leaked dataset was found to contain over a 1.4 billion pairs of emails and their corresponding passwords. To ensure that they are handling passphrases, only passwords that were reasonably long were considered. A threshold of 20 characters, or roughly 4 words (given that the average length of an English word is 4.8 characters [54]) was set, and there were around 5.7

million unique passwords satisfying the constraint. To identify the passphrase properly a segmentation algorithm was used on these passwords. First, the passwords were filtered by rejecting potential hash values [31] by using a heuristic, and even passwords with ':' symbol in the suffix and '@' in the prefix were not considered. Passwords with less than nine English alphabets were also removed. The SymSpell library [55] was used to segment the cleansed passwords with the help of a unigram distribution V (generated from popular word lists [56, 57, 58]), thereby dividing them into meaningful words. The segmented passwords were considered as passphrases only if they had 3 valid English words of at least 3 characters. From this, over 68,000 unique passphrases from the segmentation algorithm were extracted that follow the above-imposed restrictions. These passphrases were used by 72,006 users, in a total of 73,088 times. A significant portion of the users (98.4%, around 70,800 users) used only one passphrase, while some (1.6%, 1,152 users) users had more than one passphrase. Around 250 users were found to use the same password and which was considered to be the most popular, whereas 98.4% of passphrases were considered to be unique as they were used by only one user (1,082 passphrases were used by more than one user). We show some of the processed passwords from the dataset by the segmentation algorithm and the segregation of their results in various classes in Figure 4.1. '-' marks were inserted in between segmented words to highlight them. Finally, for checking the coverage of the method, we take 100 random passwords which are more than 20 characters but are discarded by the algorithm. It was manually found that 18 passwords could potentially have been considered passphrases but the algorithm failed to detect them. So the passphrase detection algorithm has a false negative rate of only 18%. By construction, it has zero false-positive rates. We will use these user-chosen passphrases (we will call it User dataset) to empirically establish the guessability and memorability results that users achieve when they are choosing the passphrases themselves.

Although these User passphrases are highly memorable, as they are designed to be so from the perspective of users, they severely fall short in the aspect of security, which is demonstrated in Section 7.2.3. Thus, to address the issue of security, users prefer system-generated passphrases over choosing their own in most scenarios. We will next see some of the most common in-use system-generated passphrases.

4.2 System-generated passphrases in use

To examine and understand different system-generated passphrases, we focus on password managers, which is one of the most popular and common avenues where system

Type	Examples

Popular passphrases	`bullet-for-my-valentine`
	`sponge-bob-square-pants`
	`correct-horse-battery-staple`

Unpopular passphrases	`friendly-realist-marking`
	`amazing-eddie-the-music-maker`
	`killer-the-best-super-killer`

Ineligible	`speed3triple123456789`
	`invalid21101975login`
	`new2jobthomasinc_sock508`

Common names	`JamesMefestoCarrasquillo`
	`zuleimahendarandez1230`
	`dobrovinelskayatatiana`

Non-phrasal	`lrdzxrfgtcyybz27112001`
	`gurtaoillocomotocsecnarf`
	`083aibgdaldnsbfatreitff`

Figure 4.1: *The figure presents the various types of passwords taken from the dataset that were processed. Some of the most common passwords are listed in the top row, while some random samples that were extracted follow in the next row. The last three rows show passwords that do not translate into passphrases along with the reasons for their failure.*

passphrases are in-use currently. We surveyed 13 popular password managers (like LastPass, 1Password, KeePass) [59]. Among these, we found that 3 of them offer to generate passphrases for the user. 1Password [60] and Enpass [61] use diceware, whereas Keepass [62] uses a template-based version of diceware as their internal algorithm to generate passphrases for users. We discuss these algorithms below:-

- **Diceware.** We see that the most common and used system passphrase generation algorithm is diceware (called Diceware from now on) [17, 14], which relies on randomly choosing words from a dictionary and combining them to make a passphrase until the required length of passphrases is achieved. Although Diceware is highly secure, the users who use the generated passphrases find it very hard to remember the passphrases, i.e., the memorability of the passphrases is very low [14]. Many approaches have tried to improve upon this and one of the most popular approaches that has achieved improvement in the aspect of memorability of Diceware passphrases is the *template based diceware* [18] or TemplateDice.

- **TemplateDice.** This can be considered an advanced version of Diceware. The algorithm has a dictionary of English words segregated based on various parts of speech tags and has 27 syntactic templates for the English language, whose components

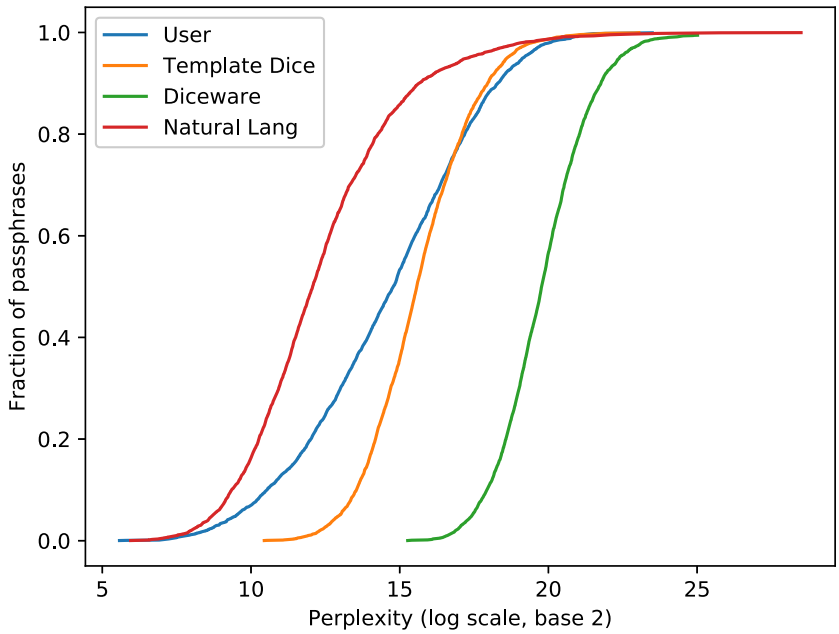

Figure 4.2: *CDF of logarithm of Perplexity to the base 2 for Diceware, TemplateDice and User passphrases, along with natural language phrases for baseline. We observe that User are the closest to the natural language with TemplateDice being almost comparable. The diceware passphrases are significantly worse in comparison.*

are the tags, embedded within the algorithm [63]. The idea of using syntactic templates has handled the issue of memorability very well, as the resulting passphrases are closer to a natural language model as opposed to the random combination of words generated by Diceware, as we see next. But, this approach has compromised on the security of the passphrases (Section 7.2.3) and sacrificed the versatility to generate arbitrary length strong passphrases as we will see next while evaluating the property of these passphrases.

4.3 Properties of in-use passphrases

In this section, amongst others, we will discuss one of the most natural properties of any passphrase, its semantic quality. Here, we demonstrate the defining characteristic of User passphrase, which is its closeness to natural language as compared to other system-generated ones. Although TemplateDice is somewhat between Diceware and User in terms of being close to natural language passphrases, it shows a plateauing guessrank with increased length—affecting the security of its passphrases.

4.3.1 Linguistic propoerties

To verify how close the passphrases generated by the various systems (and user passphrases) are to the natural language, we measure the perplexities of the passphrases from these different sources using a popular trained natural language model called GPT-2 [64]. The perplexity of a passphrase $s = p_1 p_2 \dots p_l$ is defined as $p(s)^{-1/l}$, where $p(s)$ is the probability assigned by GPT-2 model for the passphrase, which is a product of the individual probabilities of the words, and l is the number of words in the passphrase.

If given a sequence of words as an input, the GPT-2 model is trained to predict the most probable next word with respect to natural languages. Because of this, the model is widely acknowledged for its uses in the field of natural language text generation [65]. We sample around 3000 passphrases from each system (dataset for User) and compute their perplexity using GPT-2 model with the distribution of length across them being similar. Lower perplexity indicates that the passphrases behave closer to the expectations of the model, which thereby points to a better resemblance to the natural language. The distribution of the perplexity score [66] for both the system (Diceware and TemplateDice) and User passphrases, along with natural language phrases, are shown in Figure 4.2. User passphrases have a very low perplexity value which is similar to the perplexity of a natural language corpus, signifying a key reason for their memorability. On the other hand, the passphrases from TemplateDice claim a close second in their resemblance to natural language, almost comparable to user passphrases, which indicates a significant improvement over the passphrases generated by Diceware that previously performed poorly in this aspect.

4.3.2 Issue with TemplateDice

We just saw that TemplateDice has a significant improvement over diceware in its closeness to natural language. But, although TemplateDice improved upon Diceware, it comes with a compromise in security. On further investigation, we note three major shortcomings for TemplateDice. Any extension will potentially require a linguist to create new syntax rules and contextual wordlists (assuming it's possible for large passphrases).

- First, TemplateDice is not scalable as the information required by the system is all internally encoded within the algorithm [63]. If one wants to port the algorithm to a different language or would prefer an English passphrase not restricted to the templates already encoded within the algorithm, they have to change the internals by employing a linguist who has the required knowledge

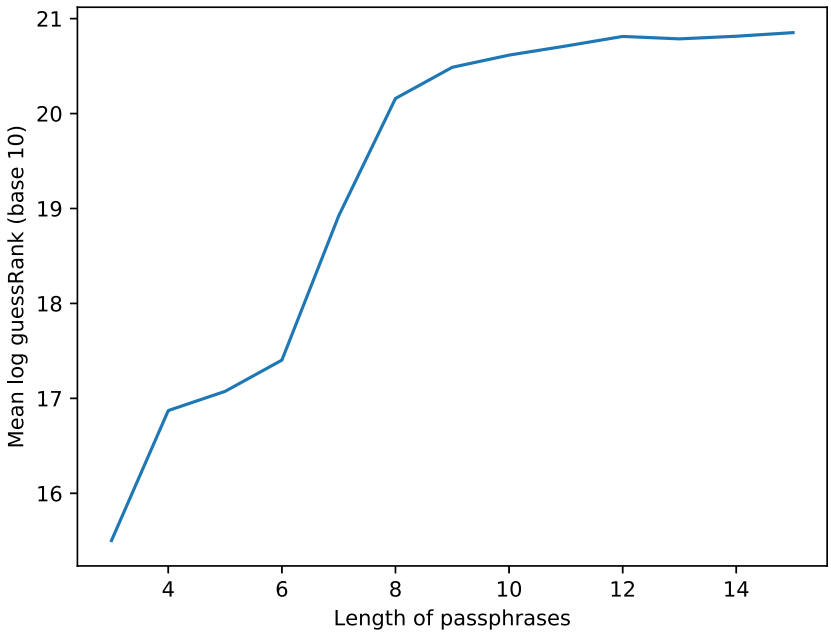

Figure 4.3: *Variation of mean $\log_{10}$ guessrank across length for the* TemplateDice *passphrases. We observe that passphrases of all lengths greater than 7 have nearly the same guessrank, which is the optimal passphrase length according to [67]. This indicates a significant compromise on the security of longer passphrases.*

- Second, and more importantly, the security of the passphrases does not scale well with length. We show the guessrank of an adversary who does brute force guess of the syntax rules and then over the wordlist of TemplateDice passphrases in Figure 4.3. We note that the guessranks of these passphrases get saturated around length 8—guessrank of an 8-word passphrase is nearly the same as that of a 13-word. But, from [67], we know that length of optimal passphrases is greater than 7. The potential reason for the fall-off in security is the constraint imposed by the underlying hardcoded and extremely limited syntax rule patterns of TemplateDice
- Another important restriction from a theoretical standpoint is that passphrases of arbitrary length cannot be generated as they are always finitely bounded by the length of the templates used.

With this insight, we aim to improve TemplateDice by resolving the above shortcomings while also not compromising on the security as well as the memorability of the passphrases. To that extent, we improve upon the MASCARA model to generate secure and memorable passphrases which address the above concerns.

Chapter 5

MASCARA

With all the necessary background and motivation, we will now present a brief overview of the original MASCARA algorithm, starting from the training data used to the generation of secure and memorable passphrases.

5.1 Training Corpus

As we know, any passphrase or password generation model needs a workable training data set, Mukherjee [11] has developed one suitable for the system's needs. The unigram and bigram Markov models that are required for training Mascara were extracted from Wikipedia as a corpus of human-generated text data. They argued that to generate memorable passphrases or measure their memorability, a good quality human-generated text corpus is needed and Wikipedia is a good source of common and shared knowledge among people in general.

They used a recent dump of Wikipedia articles [1] for the content of the corpus. The `pageviewapi` [2], the Wikimedia Pageview API client, was used to retrieve the topmost 5% of articles according to the page view count, summed up over the last half a decade. To estimate the view count accurately, the granularity of the views was set to be monthly. This only strengthens the scope of the corpus, without having to use a huge dataset, to cover all possible domains. Then the aforementioned API was employed to extract the content based on titles.

The raw content extracted from Wikipedia through the API contains numerous tags and URLs. These elements in the text do not really stand in line with what humans consider natural language and thus are not representative of them. Moreover, these elements

[1] https://dumps.wikimedia.org/enwiki/latest/
[2] https://pypi.org/project/pageviewapi/

will interfere with the evaluation methodologies devised, and hence they pre-processed the text dataset by removing all miscellaneous parts like URL and hyperlinks, captions, and HTML tags. This was achieved with the help of regular expressions, such as checking for three alphanumeric chunks delimited by '.' along with the optional presence of 'HTTP' or 'HTTPS' at the start, which then can be treated as hyperlinks. In such regular expressions, only valid extensions from the third chunk are considered [3]. Other uncommon punctuation symbols, such as ';;', '___', etc. were also removed. To make the entire corpus, all the words in the text have been converted to lower cases as prior works have shown that passphrases entirely comprised of lower case characters were easy to type [68]. Other than this, words with less than three characters, as well as any words with the presence of numeric or alphanumeric characters were also removed to keep the generated passphrases consistent with User and other system-generated passphrases. Finally, a resultant textual corpus containing 3.3 million sentences (delimited by a period '.', from around 8,210 Wikipedia articles, containing over 29 million words and 450 thousand unique words) was obtained.

We now refer to this dataset as Wiki-5, and use it as the universal corpus for the vocabulary for CER estimation and other purposes throughout the thesis and also leverage this corpus in our favor to generate secure and memorable passphrases from MASCARA.

5.1.1 Training Bigram Markov Model

The above dataset, Wiki-5, was first used to train a bigram Markov model. To do so, they added a beginning of sentence $<$ st $>$ and end of sentence $<$ en $>$ marker to each sentence identified in the dataset. Then all bigrams from such a dataset were retrieved and stored in a bi-level dictionary, counting the frequency they appear in the dataset. Words are considered a contiguous sequence of English letters delimited by non-alphabetic characters. No stemming or lemmatization of any form was carried out over the words found, as it helps to maintain the lexical distinctiveness of the generated passphrases.

We will refer to this model as $\mathcal{Z}$ henceforth, and the log probabilities of unigram and bigrams in the dataset as L_1 and L_2. Thus $L_1(p) = \log(\frac{f_p}{\sum_p f_p})$ and $L_2(p, w') = \log(\frac{f_{pp'}}{\sum_{p''} f_{pp''}})$ and all logarithms are over base 10. Note, we also use $\mathcal{Z}.\mathsf{next}(p)$ as a function that can return all the words that follow p in the corpus, using the Markov graph.

[3] https://www.regextester.com/93652

5.2 Generative Model

Here, Mukherjee in his work [11] 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 the constrained Markov generation approach as we describe below.

5.2.1 Constrained Generation

The passphrases in MASCARA are generated incrementally starting with a start symbol $< \text{st} >$. At each stage, a word is selected such that it satisfies the following constraints according to the score function derived. The score function is devised over the observation that intermediate CER values can be calculated incrementally. That is, given a partially generated passphrase $s_i = p_1 \dots p_i$, using the following equation one can compute the score (intermediate CER) value.

$$C(p_1 \dots p_i) = c_1 L_1(p_i) + c_2 L_2(p_{i-1}, p_i) + c_3 \sigma_{\text{chr}}(p_1 \dots p_i)$$

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

- $C(p_1 \dots p_i) \leq \alpha_1$ and
- $L_2(p_{i-1}, p_i) \leq \alpha_2$

  where α_1 and α_2 are two parameters of the system.

At every step of the generation, a word is randomly selected such that these two constraints must be satisfied by the intermediate passphrases. Because the score C 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 they 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 syntactic structure while being semantically meaningless.

5.2.2 The Final Algorithm

The MASCARA algorithm, which takes an incremental generation approach to produce passphrases that satisfy the constraints of memorability and guessability while trying to

GetStart($\mathcal{Z}$) :
$P \leftarrow \mathcal{Z}.\mathsf{next}(< \text{st} >) \setminus R$
$p \leftarrow_{L_1} P$
return p

MascaraGen($l, \mathcal{Z}$):
$p_1 \leftarrow \mathsf{GetStart}(\mathcal{Z})$
$i \leftarrow 2$
while $i \leq l$ do
  $P' \leftarrow \mathcal{Z}.\mathsf{next}(p_{i-1})$
  if $i \in \{1, l\}$ then $P' \leftarrow P' \setminus R$ ▷ Remove stopwords
  if $i \geq 2$ then ▷ Constraints for CER and guessrank
    $P' \leftarrow \{p \in P' \mid C(p_1 \dots p) \leq \theta_1 \text{ and } L_2(p_{i-1}, p) \leq \alpha_2\}$
  if $i = l$ then ▷ Ends in a end symbol
    $P' \leftarrow \{p \in P' \mid < \mathsf{en} > \in \mathcal{Z}.\mathsf{next}(p)\}$
  $p_i \leftarrow_{\$} P'$
  if $p_i = \perp$ or ($i = l$ and $\mathsf{isPhrase}(w_1 \dots p_l) = \mathsf{false}$) then
  if $p_i = \perp$ then
    $P \leftarrow \mathcal{Z}.\mathsf{next}(< \text{st} >) \setminus R$ ▷ No passphrase found; restart
    $p \leftarrow_{L_1} P$
  else $i \leftarrow i + 1$
return $p_1 \dots p_l$

Figure 5.1: *Mascara's constrained generation algorithm. The algorithm generates a passphrase of length l given a bigram Markov model $\mathcal{Z}$. Here R is a set of stop words, and $< st >$ and $< \mathsf{en} >$ are the start and end symbols used in the Markov model. Furthermore, $\leftarrow_{L_1} P$ denotes sampling from the support P but according to the probability distribution assigned by unigram probabilities (without log); similarly, $\leftarrow_{\$} P$ denotes sampling uniformly randomly from the elements in P.*

preserve its syntactical structure and meaning is presented in Figure 5.1.

As we sample potential words based on intermediate constraints, the approach is greedy and certainly not optimal. But, with this approach, if one plans to extend the system by expanding or changing the corpus or improving the system by modifying the system parameters, they can just do so in a straightforward manner. The current model of the system gives out a lot of potential personalizations according to user needs while still following the constraints, thus making the algorithmic approach a generalized version of producing ideal passphrases.

The choice of α_1 and α_2 are very important. MASCARA tries to ensure that, the algorithm favors rare bigrams (low α_2) while maintaining a low intermediate CER score (low α_1). Keeping this in mind, a suitable value for α_1 and α_2 is set after manually trying out a lot of combinations and checking the quality of generated passphrases. For the generation method, they set α_2 to be at least 80% of the maximum possible value of L_2 and α_1 to be 50% of the maximum possible value of L_1. These constraints ensure that the security of passphrases is not compromised by preferring rarer bigrams while CER is

also maintained.

The generation method of MASCARA is incremental ensuring the invariance of CER (memorability) and guess rank (guessability). Another way to approach the generation of passphrases keeping the same constraints in place is to generate the entirety of the passphrase of the required length lfirst and then check for the satisfaction of the various imposed constraints on the CER and guessability. Although this will ensure better results, the tradeoff in time is nowhere near worth it.

In the next chapter, we will present various optimizations and improvements over this model with the help of the previously presented background, detailing the boost in performance compared to the previous version.

Chapter 6

Proposed Work

As we have discussed before, the goal of this work is to provide a solid motivation for the development of the work based on MASCARA by Mukherjee [1], while also improving the framework and performance of the system along with the evaluation baselines. We have till now discussed the various passphrases in the wild (Section 4) and have seen the motivation for this work, and the previously presented guessability and memorability (Section 3) have improved over the original framework in MASCARA. In this chapter, we will put forward various proposals targeting the improvements to the performance of the system and the baselines. We do this in a broad range, starting from removing redundancies to introducing new algorithms for comparison.

6.1 Improving Generation Time

In this section, we will address the slow generation time of passphrases by the MASCARA system. This is done in two parts— optimizing the screening and improving the algorithm.

6.1.1 Optimising the Screening

The screening phase refers to the filtering off of passphrases by MASCARA that doesn't satisfy the required conditions, which is memorability in our scenario. As we have seen earlier, the system uses a constrained incremental generation approach 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 by the system.

As people tend to associate memorability with meaningful sentences, MASCARA also makes an approach to make the passphrases as meaningful as it can, albeit in a very straightforward way. Instead of modifying the generation process in itself, the idea used was to make a wrapper over the system so that only passphrases generated that are meaningful are considered while the rest are rejected. This means that to generate a passphrase, the algorithm is repeatedly called until a meaningful passphrase is generated. To check whether a sentence is meaningful or not, *PhraseMachine* [69] library is used.

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

To tackle the above issue, 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. This shows a very important problem. As we have used the CER values to quantify memorability so far in the work, 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 the meaningfulness of the generated passphrase. The reason the use of *PhraseMachine* is costly is primarily due to the fact that the rejection takes place at the very end, because of which a rejection results in the restart of the entire process. The same couldn't be said about using intermediate heuristics.

We will evaluate the improvements in performance offered by removing the *PhraseMachine* from the algorithm in the next chapter.

6.1.2 Improving the Algorithm

We will now be improving the actual algorithm involved in the sampling of the support, as that is the biggest bottleneck, to the passphrase generation as well as the most important.

Current Scenario. As we have discussed earlier, MASCARA uses an incremental constrained generative process to generate ideal passphrases. Its a step-by-step approach where words are chosen from a sample that adheres to certain constraints imposed by the system parameters. To be particular, given a partially generated passphrase $s_i = p_1 \dots p_i$, the next word p_{i+1} is chosen from the vocabulary such that $C(p_1 \dots p_{i+1}) \leq \alpha_1$ and $L_2(p_i, p_{i+1}) \leq \alpha_2$, where $C(p_1 \dots p_i) = c_1 L_1(p_i) + c_2 L_2(p_{i-1}, p_i) + c_3 \sigma_{\text{chr}}(p_1 \dots p_i)$. α_1 imposes a constraint on the CER, whereas α_2 restricts the guessrank indirectly to the required range. In the algorithm shown in Figure 5.1, at every step until the required word length is reached, the MASCARA takes the entire vocabulary and retrieves only those words that satisfy the required constraints with the help of the Markov model trained on Wiki-5. After which, the algorithm chooses the word p_{i+1} from the sample, weighted on their frequency of occurrence in the vocabulary. Irrespective of whether it is the filtering process or the sampling process, they execute in linear complexity on the vocabulary (or filtered vocabulary) size.

Proposed modifications. The linear complexity of the core algorithm involved in the passphrase generation of MASCARA, significantly slows down the process, considering the vocabulary size is of the order of 10^5. To rectify this, we improve the algorithm. We again do this in two parts – optimizing the filtering and improving the sampling.

Improving the complexity of the sampling of the words that follow the necessary constraints is somewhat straightforward if we have the entire support. First, a random number is chosen between $[0, 1)$. Then, we use a pre-calculated prefix array of the probabilities (cumulative distributive frequency) of the words in the support to choose a word that is closest in its CDF to the chosen random number using binary search. This changes the complexity of the sampling phase to a logarithmic scale, although a pre-processing phase whose complexity is linear on the vocabulary size is needed.

But, speeding up the filtering phase is not so easy. First, we have to do away with the $c_3 \sigma_{\text{chr}}(p_1 \dots p_{i+1})$ term, since the number of states generated to pre-process due to its presence is humongous. But the number of values this term can take is limited, and they all fit into a narrow range, and thus they can be moved over to the other side of the equation and the value of α_1 can be suitably adjusted. Now, with only the $c_1 L_1(p_{i+1})$ and $c_2 L_2(p_i, p_{i+1})$ terms to handle, the filtering of words is much easier, as they all depend only on the currently being chosen p_{i+1} and the previous word p_i. We again carry out

a linear preprocessing, where we remove all pairs of words from the dictionary that do not satisfy the constraints of unigram and bigram probabilities. When we want to choose p_{i+1}, we check the potential words that could follow p_i and pass them onto the sampling phase. Again, the preprocessing takes a linear time on the dictionary size.

With the above modifications to the algorithm, MASCARA can now generate passphrases in logarithmic complexity with a one-time preprocessing of linear time, which is a drastic improvement.

6.2 Optimising Parameter Tuning

We have previously seen that MASCARA relies on the constrained Markov generation process to produce passphrases with satisfactory CER and guessrank values. It does so with the help of the parameters α_1 (imposes a constraint on CER) and α_2 (imposes a constraint on bigram probabilities, thereby restricting Guessrank). It is not an exaggeration to say that the entire system hinges on the values these two parameters take. A low α_2 means that rarer bigrams will be chosen, while a low α_1 value implies the final CER will be less.

6.2.1 Original Approach

We will see how the system approaches the choice of parameters. It can be seen that the $C(\cdot)$ function has L_2 in it as well, so one can bound the value of C given α_2. That is to say, if $L_2(p_{i-1}, p_i) \leq \alpha_2$, then $\alpha_1 \geq C(w_1 \dots w_i) \geq c_1 L_1(p_i) + c_2\alpha_2 + c_3\sigma_{\text{chr}}(p_1 \dots p_i)$, because c_2 is negative. Thus, for a given α_1, the value of α_2 can be bounded. The σ_{chr} and L_1 terms will be at least 0, then $\alpha_1 \geq c_2\alpha_2$, or $0 \geq \alpha_2 \geq \frac{\alpha_1}{c_2}$.

From the above discussion, it is possible to narrow down the range of values α_1 and α_2 takes for good passphrase generation. The range is not optimal but works well enough in most cases. In MASCARA, the value of α_2 is set to 0.8 times the smallest value of L_2, thus effectively allowing a 20% leeway from the lowest bigram probability. With the help of the above relation and some manual tuning, a good α_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 sets of passphrases undergo manual scrutiny from multiple human reviewers and the passphrases are evaluated subjectively as well as taking 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 is needed, this method will obviously not work. Thus, we need an alternate approach that is more sophisticated.

6.2.2 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, ranking 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. The 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, 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 to 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 95%, respectively. The other 5% values of guessrank and CER will be made 0 and ∞, respectively. This will handle all anomalous cases. Note that we don't have to worry about this happening with generic passphrases because of the constraints imposed (α_1 and α_2) while generating the passphrases.

Now things have become simple. The only thing left is to decide on a range for α_1 and α_2 on which to carry out the grid search. Since α_2 imposes the restriction on the bigram probabilities, we can have the range of α_2 between some percentage of the minimum value of L_2 across all bigrams. And α_1's lower bound can be calculated with the relation discussed previously, and the upper bound can be 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 required, 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

analyzed. We notice that this approach has addressed all our previous concerns.

6.3 Improving Evaluation: TemplateDice

We have previously discussed the various passphrases in the wild currently in-use (Section 4). Among those, we have seen that user-chosen passphrases (User), Diceware system-generated passphrase (Diceware), and template-based Diceware passphrases (TemplateDice) are more common and popular. All of these classes of passphrases are commonly used for the various strengths inherent to them. From the above list, the work by Mukherjee [1] has not considered TemplateDice as a baseline for evaluation, despite it offering a significant improvement over Diceware in terms of memorability. In our work, we recalculate the results by carrying out the experiments after incorporating the TemplateDice system in the pipeline of our evaluation, with the help of the new framework of measurement. We first explain the algorithm and then present our approach for measuring its parameters for evaluation.

6.3.1 The Algorithm

This algorithm started off as a random passphrase generator until it gained popularity for generating memorable 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 is 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 establish 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 the situation. 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 of 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 are 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.

6.3.2 Guessrank Estimation

Now, we will present how we integrate TemplateDice into the guessrank framework established in Section 3.2. We have seen earlier that different cracking algorithms estimate the guessability of a passphrase differently. Currently, n-gram word (2,3) and character (4, 5, 6) are models used across all classes of passphrases as a part of estimating the guessrank. But the characteristics of the passphrase generation by TemplateDice nowhere near resemble a Markov model to just use these models to accurately arrive at the guessrank of the passphrases. Thus, we introduce a system-specific cracking model for the guessrank estimation of the passphrases of TemplateDice.

The Markov models of estimating the guessrank works even for TemplateDice, but only for passphrases of smaller length. In the case of larger passphrases, an adversary can leverage the templates and limited vocabulary in his favor and perform an exhaustive search for each template until he arrives at the passphrase. This is one of the reasons for the poor security offered by larger passphrases of TemplateDice as discussed in Section 4.3.2.

The idea is somewhat similar to a brute force approach of an exhaustive search, just done smartly. First, the adversary chooses a template from which they guess all the passphrases that belong to that class by generating them sequentially. If the correct passphrase has not been guessed, the adversary moves on to the next template. This process is repeated until a correct guess has been made. The order of the templates chosen can be random or priority-based (on the number of passphrases). As usual, the

number of guesses made is the guessrank for that passphrase.

Again, to save time and resources, we simulate the guessrank estimation process instead of actually making guesses. In this case, we assume we know which template the passphrase came from. This can be accomplished by storing the templates along with the passphrases during the generation process. Barring the fact that a phrase could originate from multiple templates, no big obstacles have to be met, as this is just an experimental scenario. Another requirement would be the approximation of the number of passphrases that can be generated from each template (each template can produce only finite passphrases). With this, we sum up the number of passphrases corresponding to each template that is not the source of the actual passphrase under consideration, that the adversary has chosen. For the template to which the passphrase belongs, a random number is used, which also contributes to the sum. The final resulting sum is the estimated guessrank for the passphrase according to this approach. This is further integrated into the *min-auto* framework (Section 3.2) along with the other cracking approaches.

In this chapter, we have seen all the modifications that have been made to the previous MASCARA. The next chapter evaluates the impact that these modifications have on the system.

Chapter 7

Evaluating the Improved MASCARA

Now that we have discussed the various changes that have been incorporated into MASCARA, we will now see what impact it leads to. In the first section, we will discuss the improvement in runtime arising from the optimization of screening as well as the algorithm. In the second section, we will evaluate the passphrases of the various systems and classes, along with the newly introduced TemplateDice, using the revamped framework of guessability and memorability.

7.1 Performance and Execution Time

We will explore the impact of the *PhraseMachine* modification in terms of running time and rejection count as well as guessrank and CER. The speedup brought about by the optimizations on the core algorithm of MASCARA is also presented.

7.1.1 *PhraseMachine*: Running Time and Rejections

So, we have removed the phrase machine. Does it do anything in terms of its execution time? Yes, it does. In fact, the observations were taken before and after showing a significant change. Since we are aiming to optimize the execution time, we will have a

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

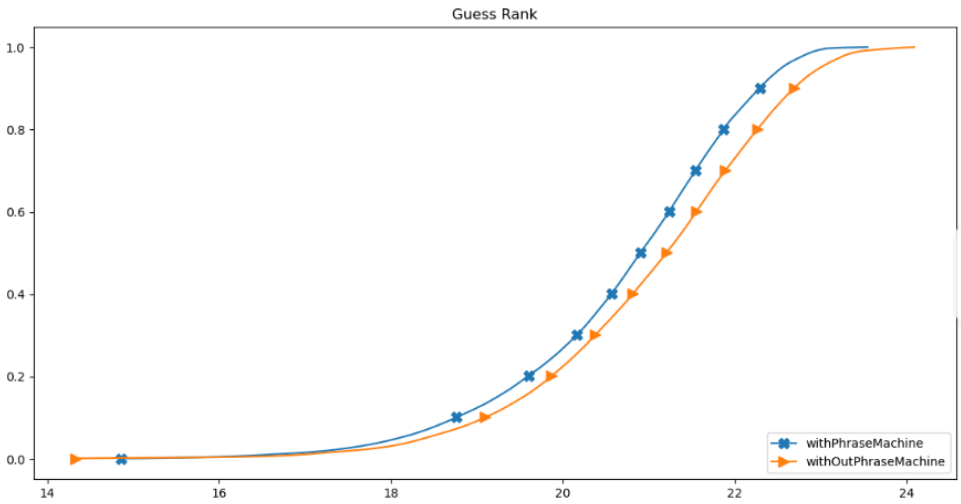

Figure 7.2: *CDF of guessrank (in log 10 scale) for passphrases of length 6 generated with PhraseMachine and without it.*

look at the figures before and after the removal of the phrase machine.

From Figure 7.1, we can see the average run time and rejection count for passphrases of different lengths calculated across 100 generated passphrases, before (X_b) and after (X_a) removing *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 in an exponential manner. This is because, the bigger the length of the passphrase, the higher the probability that at least one word of it is responsible for the rejection. Thus there is a significant improvement in terms of running time.

7.1.2 *PhraseMachine*: Guessrank and CER

But, since this is a system for generating passphrases, other than the performance of the system in terms of execution time, we also have to consider the generated passphrases. We cannot accept the change if the runtime optimization 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 7.2 and Figure 7.3. While considering both these aspects, we deem the tradeoff arising from the removal of *PhraseMachine* as significantly better than when it was included in the system.

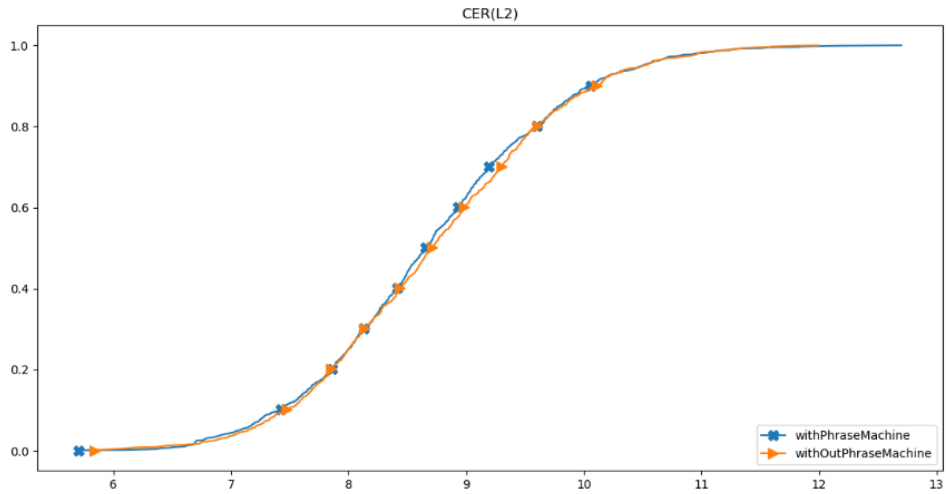

Figure 7.3: *CDF of CER for passphrases of length 6 generated with PhraseMachine and without it.*

7.1.3 Overall Execution Time

The modification to the core algorithm that has upgraded the complexity of a single passphrase generation from $O(n)$ to $O(\log n)$ is very significant, considering n is in the order of 10^5.

To evaluate the performance of the final MASCARA algorithm without *PhraseMachine* in terms of its running time, we compare it with multiple variations of itself and baselines. The baselines are Diceware and TemplateDice. The three variations are:-

- MASCARA $_{\text{end}}$: A version of MASCARA where all rejections respecting the constraints imposed by α_1 and α_2 take place only after the entire passphrase is generated by the model. This variation can be understood as a system that can create optimal passphrases for the constraints imposed.
- MASCARA $_{\text{lin}}$: The original MASCARA with the linear sampling algorithm being used.
- MASCARA $_{\log}$: The upgraded version of MASCARA where the logarithmic version of the support sampling is implemented. This is the final model implementing all modifications.

For reasonable comparisons, we keep the rest of the system parameters the same across all the variations of MASCARA.

After generating 1000 passphrases of equal length distribution across all the systems taken into consideration, we have the following observations on their execution time,

which includes the pre-processing time as well. TemplateDice and Diceware run in 7.85 seconds and 0.33 seconds, respectively. In comparison, the variations MASCARA $_{\text{lin}}$ run in 86.4 seconds, whereas MASCARA $_{\text{end}}$ takes a humongous 930 seconds. But when we compare the execution of MASCARA $_{\text{log}}$, it takes a mere 0.08 seconds, which is orders of magnitude better compared to other variations owing to its logarithmic complexity on the sample space size. This performance is even significantly better than the execution time of Diceware, and with the guarantees in security and memorability, it is even better in an overall view. From this, we can conclude that our upgraded MASCARA easily beats the previous versions, and even other systems used as baselines, highlighting the impact of the proposed work.

In the next section, we will be evaluating the quality of passphrases as just a good system performance without a promising result is not worth the user's time.

7.2 Evaluating Passphrases

The following section evaluates the quality of passphrases generated by MASCARA after the various modifications and compares it with passphrases used by users or generated using other methods using the newly established frameworks of guessability and memorability.

7.2.1 The Benchmarks

The following classes of passphrases were considered for evaluation:-

- **Diceware**. Diceware was originally proposed in [17] for generating passphrases where each word in it is randomly chosen from the vocabulary using multiple die roles. We use the wordlist that is commonly used as vocabulary in this system [19]. This class of passphrases is in general much harder to crack as they do not have any generation pattern thereof to leverage.

- **TemplateDice**. An improved version of Diceware, where passphrases are generated based on predefined syntactic templates for the English language. The templates are composed of various parts of speech like nouns, verbs, adjectives, etc., which will be replaced with suitable words from a vocabulary segregated in a similar way [63]. The passphrases generated in such a way are relatively easier to remember.

- **Markov**. We also use a bigram Markov model trained on the Wiki-5 dataset as a baseline for comparison considering that MASCARA is an enhanced version of the

former. The process of passphrase generation is similar to MASCARA. However, we don't impose any constraints on the intermediate steps and sample words weighted on their conditional bigram probability.

- **User**. We use the previously identified large number of user-created passphrases from prior password leaks (Section 4.1). These passphrases are manually chosen by users, hence should be easy to remember as they will have a certain structure to them.

- **Mascara**. Finally, we consider the improved MASCARA model, which also internally uses a bigram Markov model trained on the Wiki-5 dataset, with the several control parameters optimized by a grid search to ensure the generated passphrase has higher memorability while maintaining a high guessrank (Chapter 5).

Random samples of passphrases from each of the aforementioned classes are shown in Figure 7.4.

7.2.2 Test Samples

For the purpose of evaluating the different sets of passphrases, we need a test suite. For that, we generated 1M passphrases from TemplateDice, and following the same distribution of lengths, we generated 1M passphrases from each of Diceware, Markov, and MASCARA. In the case of User, owing to the limited size of the dataset, we use only 6,500 user passphrases as our test sample. The results are shown in terms of their CDF, and hence the different test sample sizes will not introduce any significant bias. Moreover, the length distribution of these User passphrases is different from those used above, as users often tend to utilize passphrases of smaller lengths. We didn't use the distribution of User for the system generated samples because we believe that all the entities are best evaluated at their natural state, and this implies letting User have passphrases of smaller length and allowing the systems to generate passphrases as they would normally.

7.2.3 Security of the Passphrases

We measure the strength of a passphrase based on their guessrank using the *min auto* approach discussed in Section 3.2. To compute the *min-auto* guess rank of password we take the minimum guessrank according to a number of guessrank estimations.

We consider seven different guessing algorithms for the *min-auto* approach: 2-gram and 3-gram word-based Markov models and 4-gram, 5-gram, 6-gram character-based

Type	Samples
Diceware	valedictory silliest illeism niglichen dreamscape manchuria dervish verbally whinging ferries portmore permanency downcast epilogue guarding richemont feeney stoppers scalable deuteronomic kudo
TemplateDice	a flea will republish your gladiator how does its 1 shire milk a jack the frothy thing faxed the aphorism the rivalry was misspelling prior to our eyeballs can Carole wrestle the gaiter
Markov	murrays situation spores have protractable and the problem standard kit during ultraviolet signals beamed there improvements achieved worldwide with coitus are shipwrecked man
User	little bitty pretty one dont forget the password just another happy ending ride with the wind hot and sexy chicken wing
Mascara	woolwich the fleet including queen many local fiscal rights lee sung before god jackal with prayer requests drakes book nearly too quickly

Figure 7.4: *Five randomly sampled passphrases from each group of passphrases we consider for evaluation.*

Markov models [70], template-based guessing (Section 6.3.2) and Wiki-5 bigram model. We explain the last guessing model below.

Probabilistic Wiki-5 bigram Markov. For Markov and Mascara, other than training on a huge corpus of generated passphrases, an attacker can also train it on the dataset using which the passphrases are generated, namely Wiki-5. As a bigram Markov model is used for the generation of the passphrases in these two systems, we also use a probabilistic bigram Markov guessing model trained on the Wiki-5 dataset.

While the word and character Markov models are used for the guessrank estimation of all sets of passphrases, template-based guessing and Wiki-5 bigram model are class-specific. Template-based guessing is used for the passphrases of TemplateDice, whereas Wiki-5 bigram Markov guessing model is used for the passphrases from Markov and Mascara.

Recall that according to the threat model described in Section 2.2, the attacker has access to a huge corpus of passphrases generated from the system whose passphrases are being cracked. Therefore, the attacker has 10^7 system-generated passphrases from

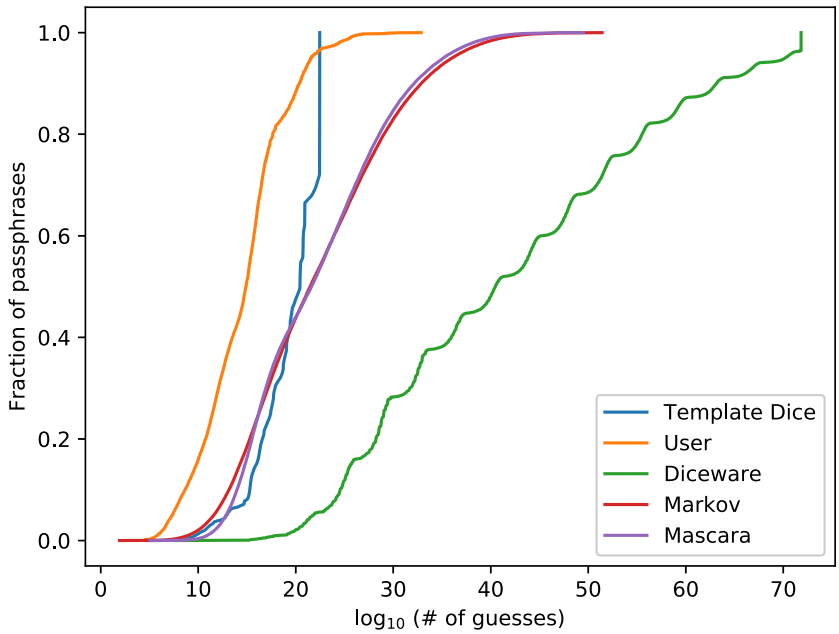

Figure 7.5: *Comparing the strength of different sets of passphrases by evaluating their guessrank. Here, a cumulative distribution frequency of log guessrank (base 10) is shown. We can see that Diceware passphrases are the most secure, while the User ones are the most predictable.*

{Diceware, Markov, Mascara, TemplateDice }, on which the attacker can train his word and character-based Markov models. In the case of User passphrases, the attacker trains on a smaller set of 70 thousand passphrases. This trained model is then used to guess the passphrases of the corresponding test samples. Each passphrase has a probability of generation according to a model and using the method in Section **??**, we can estimate the guessrank that is, the number of guesses the attacker will need to guess the passphrase correctly using that particular guessing model. A smoothing factor is used for any out of vocabulary (OOV) n-grams encountered.

The minimum guessrank across all the common models and system-specific models is then calculated and taken as the final guessrank for each passphrase.

Results. We show the (estimated) guessranks of the passphrases in the test samples in Figure 7.5. Diceware passphrases are the most secure with 50% of passphrases requiring at least 10^{40} guesses. On the other end of the spectrum, we have User passphrases, with 50% of passphrases guessed within 10^{14} guesses, which is not even as secure as some of the most secure passwords [20]. The predictability of User passphrases can be somewhat attributed to their smaller length, but since that is the inherent nature of these passphrases, we did not see fit to change it. In between User and Diceware, we have Mascara, Markov, and

TemplateDice. The security of both Markov and Mascara are similar, with Mascara having a slight advantage over Markov in the 20% most predictable passphrases of each set.

The main aim of Mascara is to resolve the shortcoming of TemplateDice. Comparing these two, we see that the only advantage the latter has over the former is that the guessrank of TemplateDice is slightly higher than Mascara for passphrases below the 40^{th} percentile. These are the passphrases that are of a length less than or equal to 8. As we move along the curve, we observe a huge difference in the number of guesses needed by Mascara and TemplateDice for their most secure passphrases. TemplateDice needs 10^{22} guesses for at least 20% of the passphrases, whereas Mascara significantly improves upon it and requires over 10^{30} guesses for the same.

However, in the next section, we discuss how memorable are the passphrases from different sources, as they are another important aspect that must be taken into consideration.

7.2.4 Memorability of Passphrases

For ideal passphrases, they have to be both hard to guess as well as memorable, so that they can be used in practice. In this section, we evaluate the memorability of the passphrases of different classes in the test suite based on the memorability framework established previously in Section 3.1.

Results. The distribution of CER of the passphrases is shown in Figure 7.6. As expected, the position of the highest CER, which means a low memorability, is attributed to the system-generated passphrases by the Diceware. A good 50% of the passphrases have a CER of more than 17%, which means that as users type in every 6 characters, they are very likely to make at least one mistake in some form. We hypothesize the lack of any syntactic structure is responsible for such high CER. User passphrases seem to perform the best, with 80% of the passphrases with less than 5% CER (a mistake every 20 characters). This is within expectations, given that users, in general, choose highly common phrases, quotes, and song or movie titles.

CER values of passphrases from Mascara, Markov, and TemplateDice are in between that of User and Diceware passphrases. All the three CERs are almost similar (with TemplateDice having a slight advantage), with each of them having a 12.5% CER for at least 50% of the passphrases in their corresponding samples— significantly better than Diceware, although much worse than User passphrases.

The results discussed above show that Mascara has overcome the limitations of TemplateDice discussed in Section 4.3.1 and is much more effective to use in practical scenar-

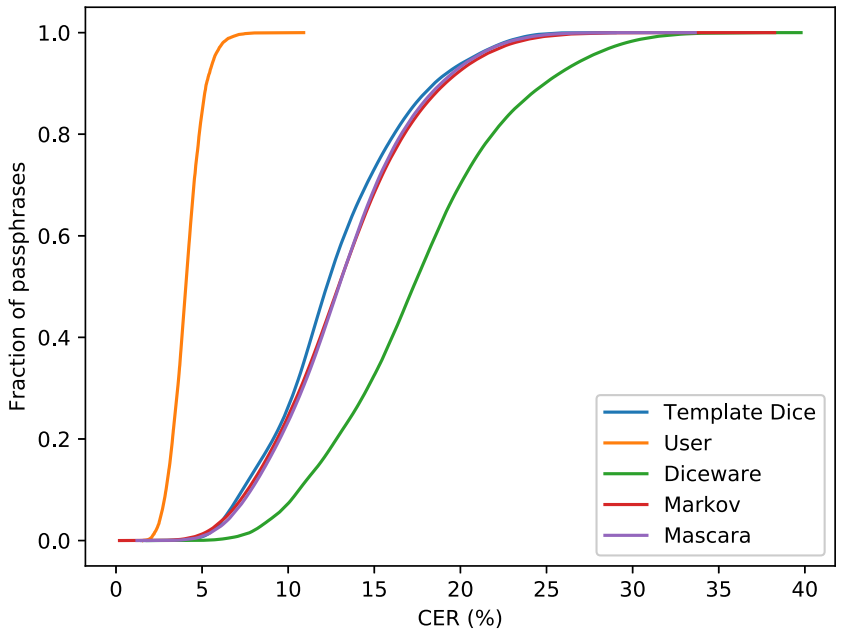

Figure 7.6: *Distribution (CDF) of CER (Character Error Rate) for passphrases among the various samples. Diceware passphrases have the highest CER, making them the least memorable, and User passphrases have the lowest CER which in turn means that they are easiest to remember.*

ios. This is further corroborated by the μ value (which was discussed in Section 6.2.2) comparisons between Mascara and TemplateDice shown in Figure 7.7.

Although Diceware offers significantly high security, users will also find it hard to remember, owing to its high CER. On the other end of the spectrum, while User passphrases are very easy to remember, they offer little to no security. Furthermore, given the parameterized generation process of Mascara, one can configure it to a setting that meets their needs, giving room to a lot of personalizations.

7.2.5 User Study

We used CER to estimate the difficulty of memorizing various passphrases analytically. However, to evaluate if indeed, final MASCARA generated passphrases are memorable to the users, a user study was carried out.

The two-part longitudinal survey-based user study was deployed on the crowdsourcing platform Prolific where it was assigned Prolific participants at random to one of the passphrase generation algorithms out of TemplateDice, Mascara, Markov, Diceware and show them a passphrase to remember. For each algorithm, the passphrases were ran-

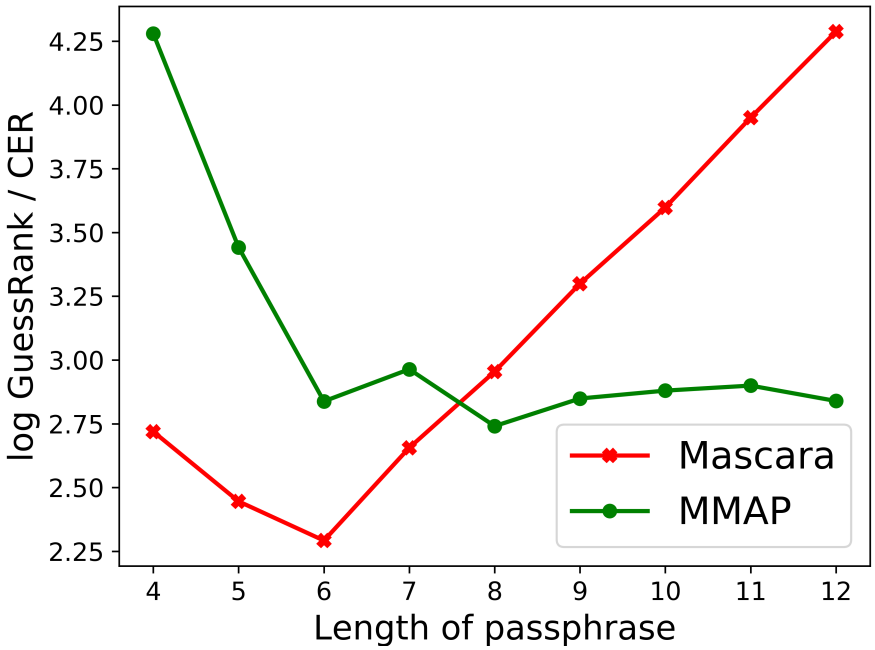

Figure 7.7: *Average ratio of log guessrank and CER (μ) of passphrases of different lengths generated by Mascara and MMAP.*

domly generated while uniformly choosing passphrase length from one of three length ranges ($\leq$ 7, 8-12, >12). The range of $\leq$ 7 was used as most of User lies in the range, while the other brackets corroborate the fact that passphrases of length greater than 7 are the most suitable [67]. The study was designed based on prior work on measuring memorability of login credentials [27].

In the first part, each participant was asked to choose from the three passphrases shown to them, all of which were generated by the same algorithm (randomly chosen for the participant) and were of the same length. They were made to practice their chosen passphrase five times after finalizing their choice. Then, each participant was asked to authenticate twice - at the end of part 1 of the survey, and the beginning of part 2, which was carried out after a two-day recall time interval. This period is used for most prior password and passphrase recall research [27, 39, 29]. We allowed at most five tries to authenticate in both parts of the survey. The users were asked not to paste their answers and were also assured that they would receive payment regardless of their authentication success. To give participants a distraction before asking for authentication, the participants were made to answer demographics and some generic questions as well as attention check questions. At the end of the authentication in the second part, participants were provided a short survey to assess their perception of the chosen passphrase and how

Model	Recall	Mean CER	Median CER
Mascara	26.23%	34.78%	35.85%
TemplateDice	17.46%	35.44%	36.58%
Markov	21.95%	37.84%	41.27%
Diceware	24.00%	38.49%	42.57%

Figure 7.8: *% participants with successful recall, mean and median CER (as %) while authenticating after 2 days.Mascara perform best. Surprisingly Diceware is close second.*

they may want to modify the passphrase.

Recall and CER. In this case, a *successful recall* and *low CER* signify high memorability. The recall is successful if in part 2 users correctly input every character (including spaces) of the passphrase within five attempts. Also, the character error rate (CER) for a particular attempt is the edit distance between the attempt and the original passphrase divided by the number of characters in the original passphrase. The CER is calculated for a user as the minimum CER across all attempts.

In part 1 (5 minutes after seeing the passphrase), Mascara, TemplateDice, Markov, and Diceware have recall rates of 59.72%, 51.28%, 63.76%, and 56.07%, respectively. The recall rates of the algorithm are very similar to each other, signifying that in short term, passphrase algorithms do not have a significant impact on the memorability of the passphrases.

Recall after two days. Figure 7.8 shows the percentage of users who were able to successfully recall passphrases chosen in part 1, as well as the mean and median CER for all users during the authentication after 2 days in part 2 of the survey. Mascara has the highest recall rate of 26.23% among all algorithms (TemplateDice recall only 17.46%). Surprisingly, the recall rate of Diceware is 24%, comparable to Mascara.

Two potential key reasons were identified which helped Diceware recall. First, the users' self-reported reasons in the survey indicated for 41.46% users Diceware contains frequently used words and it is the top reason behind Diceware memorability. Second, the return rate of Diceware participants for part 2 was low, only 60.97% (compared to 84.72% for Mascara). So, potentially, only users who were reasonably confident about successfully recalling passphrases returned in part 2, biasing the recall. However, for the same Diceware users the median CER is highest, showing that users error significantly more (i.e., cannot remember properly) for Diceware generated passphrases (compared to Mascara) when they could not recall a passphrase perfectly, affecting the utility of Diceware.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this work, we have taken a big step forward towards the goal of generating extremely secure and memorable passphrases by improving upon the previously proposed MASCARA system by Mukherjee [1]. We have seen 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 concerns.

We begin with the framework of guessability and memorability developed by Mukherjee [1]. These address the need for every passphrase to require some form of evaluation for its security and ability to be remembered, as well as its use in the core algorithm of passphrase generation. The previous guessability framework, which although used guessrank as a foundation and developed in the correct direction, is not able to face off against the sophisticated cracking mechanisms of modern attackers. Hence, changes have been proposed in this work based on previously established systems, which use the *min-auto* approach by taking the minimum guessrank across multiple cracking algorithms to address the issue of real-world security. In the memorability section, although the system developed is solid, it doesn't corroborate with practical results. To that extent, we carry out experiments on phrases tagged with CER values to establish the correlation between the various factors of a passphrase and the character error rate.

Furthermore, we have also explored the various classes of passphrases currently in use, comprising user-chosen User passphrases and automated system-generated passphrases.

8. CONCLUSION AND FUTURE WORK

The User passphrases were analyzed with the help of the foundations laid down in the work by Mukherjee [1] that has retrieved a large number of passphrases from a password database leak. It was also shown that such passphrases are not secure enough. Then we moved on to the class of system-generated passphrases, where we carried out a survey of all popular password managers. From there we found that Diceware is a commonly used system, but suffers from the drawback of low memorability despite its high security. Rectifying this, TemplateDice was developed, but this further comes with issues of its own like its lack of versatility and extendability, a saturation of guessrank, and limited passphrase length. To address these concerns, the core of the MASCARA has to be improved, which is the focus of this work.

We start off with the performance of the MASCARA in terms of its execution time. The original system was extremely slow in terms of the generation of passphrases, owing to various bottlenecks. The most redundant of them all is the usage of *PhraseMachine* as a part of the linguistic check to ensure the generation of meaningful passphrases. But passphrases do not have to be linguistically sound for them to be easy to remember, leading to its removal from the algorithm, resulting in a speedup of its generation by over 25 times.

We also carry out another significant modification to the core algorithm of the system. At every step, the algorithm chooses a possible set of words that follow the constraints imposed on the vocabulary and then samples them according to their frequency. All these happen linearly in an inefficient manner. We have modified this by performing a bit of preprocessing, and making the actual passphrase generation complexity to a logarithmic scale, resulting in another significant speed of nearly 1000 times, making the difference in performance obvious.

Moving on, 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. Owing to the fact that this is not feasible when finer precision on parameter values is needed, or if additional parameters are being introduced in the future, we have developed an evaluation metric for the generated passphrases that can then be used to automate the search for optimal parameters by performing a grid search and ranking the system corresponding to the generated passphrases of each different value. This addresses all concerns arising from human intervention.

Finally, we evaluate the quality of passphrases generated by the modified MASCARA system, in comparison with some of the old baselines, along with the newly introduced TemplateDice. We have used the revamped framework of guessability and memorability

for the evaluation being carried out. Through this evaluation, we have revealed that the passphrases generated by MASCARA have a proper systematic tradeoff between memorability and guessability, beating other classes of passphrases that have a focus on only one of them. It has also overcome the various shortcoming of TemplateDice discussed. A user study was also carried out showcasing the high memorability of MASCARA generated passphrases, and their low mean and median CER when used by the participants of the study.

8.2 Future Work

Even with the almost complete overhaul of the MASCARA by carrying out the various modification proposed, we feel that the system still has lots of potential left uncovered. If we did carry out the further enhancements, the system is likely to undergo another major improvement. Some promising avenues that could be targeted are:-

- One of the most important parts of the MASCARA system is the training corpus used. The currently existing dataset is reasonably good with a good focus on general user trends, 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 a certain bigram is only once, and such bigrams aren't exactly uncommon. It is obvious that such tails will hamper the resultant Markov model's generation process. So, some ideal way of tackling this issue has to be identified, or new datasets have to be explored.

- We have removed the use of *PhraseMachine*, which tests for linguistic correctness of the final generated passphrase and have replaced it with just a simple heuristic of just testing for stopwords at the start and end of the passphrases. There are a lot of ways where we can almost imitate the effects of the usage of the *PhraseMachine* while still not losing the performance improvement we have achieved. One way could be to introduce more powerful heuristics into the system that works on the intermediate instead of directly on the final passphrases, like guiding the incremental addition with PoS (parts of speech) tags.

- Another addition that could potentially improve the system a lot is involving trigrams, although the effects have to be experimented with before they could be confirmed. Involving trigrams can potentially not only increase the accuracy of the estimation of guessability and memorability, but they can also provide more control over the generation process. But, incorporating the trigrams into the system

must be handled with care because of the significant overhead it might have on the generation process, which we have painstakingly addressed in this work.

- Testing the versatility of the MASCARA system can allow one to understand the potential use cases it can be involved in. Furthermore, in TemplateDice, we noted that one of the major drawbacks is that it cannot be extended for other scenarios like different languages, etc. But, in MASCARA, no part of the overall system is language-specific, maybe except for the memorability, even which could be recalibrated with the help of a corresponding training dataset for the memorability framework. After that, we can test for the generation of passphrases from other languages and even multilingual passphrases might not be too big an obstacle.
- Finally, and most importantly, actually improving the memorability of the generated passphrases even further without giving away in terms of security will make the passphrases significantly more useful. This is because, we have seen from both the experiments and the user study, although MASCARA has an advantage over other classes of passphrases, it might not be significant enough.

References

[1] Avirup Mukherjee. Systematic generation of secure and memorable passphrases, 2021.

[2] 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.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] Mark Keith, Benjamin Shao, and Paul Steinbart. A behavioral analysis of passphrase design and effectiveness. *Journal of the Association for Information Systems*, 10:63–89, 02 2009.

[9] Yishay Spector and Jacob Ginzberg. Pass-sentence a new approach to computer code. *Computers & Security*, 13(2):145–160, 1994.

[10] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: empirical results. *IEEE Security Privacy*, 2(5):25–31, 2004.

[11] Moshe Zviran and William J. Haga. A comparison of password techniques for multilevel authentication mechanisms. *Comput. J.*, 36:227–237, 1993.

[12] 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.

[13] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, 2012.

[14] 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.

[15] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE, 2012.

[16] Noopa Jagadeesh and Miguel Vargas Martin. Alice in passphraseland: Assessing the memorability of familiar vocabularies for system-assigned passphrases, 2021.

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

[18] Make me a password. https://makemeapassword.ligos.net/. Accessed: 2022-01-31.

[19] Joseph Bonneau. Eff's new wordlists for random passphrases, Feb 2018.

[20] Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. Measuring Real-World accuracies and biases in modeling password guessability. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 463–481, Washington, D.C., August 2015. USENIX Association.

[21] 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.

[22] 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*, pages 1569–1575, 2015.

[23] 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.

[24] 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.

[25] Yi Liu, Ruilin Li, Xingtong Liu, Jian Wang, Lei Zhang, Chaojing Tang, and Hongyan Kang. An efficient method to enhance bitcoin wallet security. In *2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 26–29, 2017.

[26] 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.

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

[28] 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.

[29] Zeinab Joudaki, Julie Thorpe, and Miguel Vargas Martin. Reinforcing system-assigned passphrases through implicit learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1533–1548, 2018.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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)*, pages 65–80. USENIX Association, 2012.

[36] 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.

[37] 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.

[38] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 162–175, New York, NY, USA, 2010. Association for Computing Machinery.

[39] Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy M. Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio López Hernandez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. *2012 IEEE Symposium on Security and Privacy*, pages 523–537, 2012.

[40] Joseph Bonneau. Statistical metrics for individual password strength. In *Security Protocols Workshop*, 2012.

[41] 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.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

REFERENCES

[49] E. L. Lehmann and Joseph P. Romano. *Testing statistical hypotheses*. Springer Texts in Statistics. Springer, third edition, 2005.

[50] Marcel Adam Just and Patricia A. Carpenter. A theory of reading: from eye fixations to comprehension. *Psychological review*, 87 4:329–54, 1980.

[51] 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.

[52] 1.4 billion clear text credentials discovered in a single database. https://bit.ly/3r512M7. Accessed: 2021-01-28.

[53] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for checking compromised credentials. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1387–1403, 2019.

[54] P. norvig, english letter frequency counts: Mayzner revisited or etaoin srhldcu. http://norvig.com/mayzner.html. Accessed: 2021-01-28.

[55] Wolf Garbe. Symspell, 2019.

[56] Chapter 02: Natural language processing with python, by steven bird, ewan klein and edward loper. http://www.nltk.org/book/ch02.html. Accessed: 2021-01-28.

[57] Major cities of the world. http://www.geonames.org. Accessed: 2021-01-28.

[58] Moby word lists by grady ward. http://www.gutenberg.org/ebooks/3201. Accessed: 2021-01-28.

[59] Survey of password managers. https://bit.ly/3ukEWKf. Accessed: 2022-01-31.

[60] 1password passphrase generation. https://github.com/1Password/spg. Accessed: 2022-01-31.

[61] Enpass passphrase generation. https://www.enpass.io/password-generator/. Accessed: 2022-01-31.

[62] Keepass passphrase generation. https://keepass.info/plugins.html#ppgen. Accessed: 2022-01-31.

[63] Template based diceware algorithm. https://github.com/ligos/MakeMeAPassword. Accessed: 2022-01-31.

[64] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[65] Better language models and their implications. https://openai.com/blog/better-language-models/. Accessed: 2021-01-28.

[66] Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. An estimate of an upper bound for the entropy of english. *Comput. Linguist.*, 18(1):3140, mar 1992.

[67] Christopher Bonk, Zach Parish, Julie Thorpe, and Amirali Salehi-Abari. Long passphrases: Potentials and limits. *CoRR*, abs/2110.08971, 2021.

[68] 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.

[69] 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.

[70] Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy*, pages 689–704, 2014.