`record from the adapter`: On Systematic Generation of Memorable And Secure Passphrases by MASCARA

Anonymous Authors

Abstract

Passwords are the most common mechanism for authenticating users online. However, studies have shown that users find it difficult to create and manage secure passwords. To that end, passphrases are often recommended as a usable alternative to passwords, which would potentially be easy to remember and hard to guess. However, as we show, user-chosen passphrases fall short of being secure as they prefer those that are easy to remember, and state-of-the-art machine-generated passphrases are difficult to memorise.

In this work, we aim to tackle the drawbacks of the state-of-the-art systems that generate passphrases for practical use. In particular, we address the problem of generating secure and memorable passphrases and compare them against user chosen passphrases in use. To that extent, we identify and characterize 73,044 user-chosen in-use unique English passphrases from prior leaked password databases. Then we leverage this understanding to create a novel framework for measuring memorability and guessability of passphrases. Utilizing our framework, we design MASCARA, which follows a constrained Markov generation process to create passphrases that optimize for both memorability and guessability. Our evaluation of passphrases, shows that MASCARA improves the security of passphrases over in-use user-generated passphrases, and has none of the shortcomings found in the system generated passphrases in use. Furthermore, our user study shows that, users have the highest recall rate for passphrases in their preferred range with MASCARA performing 61.53% better than the next best model.

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, 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 [61]. To that end, *Passphrases* are considered an alternative approach to generate memorable yet strong authentication secrets. Passphrases are a sequence of words (e.g., "correct horse battery staple") as opposed to passwords which are a sequence of characters.

Passphrases are often recommended for particularly sensitive cases like the master secret for password managers [33], to lock their Cryptocurrency wallets (such as brainwallet [27]) or even protecting ssh keys [11]. Passphrases could also be used for creating and remembering more memorable passwords as well (e.g., a mnemonic created from a passphrase [64] can be used as a password with improved memorability).

In this work, we 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 text [37, 38, 55, 65, 66] however they are also relatively easily guessable due to this very structure [16, 21], thus also increasing their chance of getting attacked. Specifically, earlier works [21, 54] also found that user-generated passphrases are memorable yet weak which an attacker can comfortably guess. To that end, system-generated passphrases are proposed—their security advantages include resistance to targeted impersonation, throttled guessing, unthrottled guessing, and leaks from other verifiers [19]. Unfortunately, the use of system-assigned passphrases comes at a cost to memorability [34, 54].

Specifically, one of the current systems that generate passphrases for practical use is Diceware [52, 54], which involves choosing random words from a given wordlist. However, Shay et al. [54] showed that passphrases generated by this approach are error-prone for users and the resulting passphrases are not memorable. Their memorability was similar to randomly generated passwords.

We found that another variation of diceware, *Template based diceware* is used in the wild [8]. 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 usage of a handful of specific syntax templates and (ii) need for non-systematic manual effort to extend passphrase generation with more templates (Section 3.3), necessitating a thorough re-design of the approach. Several works took an alternative approach—they tried to use various learning techniques like implicit learning post-passphrase generation to improve their memorability [34]. 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 answer this question affirmatively and present MASCARA, which can automatically generate memorable passphrases with good security. We design MASCARA using insights from a novel in-use English passphrase dataset to ensure good guessability-memorability trade-offs. 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 [18, 21, 34]. We leave extending the system to other languages as future work.

The key contributions of this work are:

- We created a novel organic dataset of 73,044 user-chosen unique English passphrases using a simple algorithm on password leak databases. To the best of our knowledge, our dataset is the largest user-generated passphrase dataset to date [1]. Our algorithm leverages word segmentation in the noisy text to identify these passphrases. We found that the syntactic structures of these passphrases are distinctly different from Diceware, which likely enhances the memorability of passphrases, however, possibly also increasing the guessability of passphrases.
- We have created a memorability-guessability measurement framework for passphrases. Using prior works on the memorability of natural language phrases and the experiments we carried out, we identified distinct and important features of a sentence (e.g. frequency of occurrence, the standard deviation of character length of words, etc.) which affects the memorability of a passphrase. We additionally created a Monte-Carlo estimate of the guess ranks of passphrases which measures the guessability of passphrases (according to prior work in this space higher guessability implies lower guessrank). We utilize this framework to balance memorability and guessability during passphrase generation.
- Using our framework, we present MASCARA, a novel system to automatically generate memorable yet not so easily guessable passphrases. MASCARA leverages a constrained generative process by modifying a generative Markov model and explicitly considering the dimensions of memorability and guessability during passphrase generation. Our evaluation demonstrated that MASCARA generated passphrases have improved security than user-generated ones and do not suffer from any of the drawbacks of the existing systems. Moreover, the user study we carried out, shows that users have the highest recall rate after two days for passphrases in their preferred range with MASCARA performing 61.53% better than the next best model. The user study also shows that MASCARA also has the lowest median character error rate or CER (35.85%, Section 4.1) compared to other system-generated passphrases in the wild (with over 42% CER).

The rest of the paper is organized as follows: We provide background on passphrases in Section 2. Then we explore the current passphrases in use, both user-chosen (by extracting passphrases from password leaks) and system generated in Section 3. Section 4 provides our memorability-guessability measurement framework and consequent design of MASCARA. Finally Section 5 and Section 6 evaluates the usability of MASCARA-generated passphrases in comparison to other baselines and Section 7 concludes this work.

2 Background and related work

We first provide some background on current work on system-generated passphrases and then our threat model. Finally, we provide background on the memorability of the authentication mechanisms with a focus on passphrases.

2.1 System-generated Passphrases

Many earlier works focused on passwords as an authentication mechanism for critical as well as non-critical infrastructures [19]. Unfortunately, earlier works also demonstrated that users tend to often choose predictable passwords and reuse the same passwords across multiple accounts [25, 30]. Consequently, multiple proposals aimed to improve the security of authentication mechanisms, by designing better password meters or creating mnemonic passwords [42].

To that end, in the past decade, passphrases are being put forward as an alternative or complementary mechanism to passwords. NIST defined a passphrase to be a *memorized secret consisting of a sequence of words or other text that a claimant uses to authenticate their identity* [31]. Intuitively, we can see that passphrases are likely to be easier to remember than passwords (due to their closeness to natural language) for users as well as harder to guess (due to their length) for adversaries. Shay et al. [54] demonstrated

[1] We will release code and data for this dataset in the public domain at the time of publication.

(using a user study) that even simply using passphrases comprising of three to four words can be comparable in terms of entropy with passwords generated using more-involved methods while also accounting for memorability, highlighting the utility of good passphrases. In the current scenario, passphrases (especially system-generated ones) are quite common, and used in password managers, cryptocurrency wallets [46], and ssh-keygen [11].

We note that the earlier works on password creation often focused on generating secure and memorable *passwords* over the years. They used techniques like contextual cues, portmanteau, or mnemonic based generation [14, 42, 56, 63]. These techniques to improve memorability often aim to associate a context (like a memorable phrase) associated with the password. Consequently, our work on secure and memorable passphrase generation is complementary to the generation of memorable passwords—a secure and memorable passphrase can easily be leveraged to create a secure password (while using the passphrase as context for memorability).

However, earlier studies and works have not systematically investigated the guessability-memorability trade-off using a framework and leveraged it to create memorable system-generated passphrases. In this work, we rectify the above issue by formalising the guessability-memorability problem and we do so through leveraging a well-defined data-driven framework. Our approach is complementary to the works on passphrase learning approaches which aim to help users learn and memorize system-generated passphrases [35]. 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

Measuring security of passwords is a well-studied topic [40]. Earlier works considered different approaches the attacker could employ and thus used Markov models, probabilistic CFG, neural networks, etc. for the guessability estimations [49, 50, 57, 58, 60, 61]. However, there is relatively less work in the domain of passphrases.

Using guessrank to measure passphrase security Previous works found that the security of passphrases can be increased by increasing the entropy via either introducing semantic noises or increasing the wordlist size [43]. However, if users are given control, they generally tend to opt for common phrases, reducing security drastically [42]. In all of these works, the security of passphrases is generally measured through either entropy or user-based surveys [53].

But later, researchers have shown that *guessrank* is a much more acceptable measure than entropy for measuring the security of a password [59, 62]. The Guessrank of a password can be understood as the number of guesses an adversary needs to arrive at the correct password. So higher the guessrank, lower the guessability. Building upon this and earlier works on password meters [49, 57, 58, 60], we use the guessrank metric for measuring the guessability of passphrases in our setup. Specifically, we estimate a guessrank for each passphrase in our setup— the higher the guessrank, the higher is its resistance to external attacks.

Our adversary model with *min auto* approach In this work, we consider a powerful offline generalized untargeted adversary. In our threat model, the adversary knows the exact corpus from which the systems are generating passphrases and the exact algorithm of the passphrase generation. 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. We assume that the attacker also has access to a huge dataset (e.g., over 10M) of passphrases generated by live systems. The primary challenge for the attacker is to generate an ordered list of passphrase guesses $\omega_1, \omega_2, ..., \omega_n$ to reach the target passphrase ω as early as possible (least number of guesses). The higher the guessrank of a passphrase the stronger the passphrase is (lower guessability). Given there are multiple algorithms an attacker can use to guess a passphrase we take a *min auto* approach described by Ur et al. [59]. First, we used multiple passwords 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 [17, 39]. For each algorithm, the guessrank will be the number of incorrect guesses the particular algorithm used to arrive at the correct passphrase. Since the average guessrank for even a passphrase of medium strength is of the order of 10^{15} (Section 5.1), actually running the algorithms to find the guessrank is infeasible—we simulate the running of the algorithm using Monte Carlo simulations for our purpose (Section 4.2). Finally, following the previous work by Ur et al. we simply took 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 scenario [59].

2.3 Measuring memorability of passphrases

Earlier works tried to correlate password memorability with the frequency of passwords, login durations, and even keyboard pattern [28, 32]. Other works used methods like encoding random n-bit strings for generating memorable passwords or using chunks [20, 30]. All of these cases measured memorability of passwords based on user surveys instead of an automated linguistic metric [23, 38, 54, 63]. Although useful, these works considering memorability of

passwords are complementary from that of passphrases. Passphrases often contain possible linguistic properties (the order in which words are presented) which are generally absent in passwords. To that end, there is some work on the memorability of English phrases. For example, work by Danescu et. al. [24] has tried to measure memorability of popular movie quotes using lexical distinctiveness. Some Human-Computer Interaction studies identified Character Error Rate (CER) as an acceptable measure for memorability of passphrases [41, 48]. We build on this research, identify underlying factors affecting memorability of phrases, and consequently optimize to improve memorability of MASCARA-generated passphrases (Section 4.1 and Section 4.4). Our longitudinal user study results demonstrated that users indeed memorize MASCARA generated passphrases which leverages this model. With this background, in the next section, we start with first identifying and analyzing the state of the art methods for generating passphrases in the wild.

3 Understanding Passphrase generation in the wild

Today passphrase generation falls into two classes—user-generated and system-generated. User-generated passphrases (or *User* passphrases) are picked by users and potentially easier to remember for them. System-generated passphrases, are algorithm-generated passphrases (often parameterized by a training corpus or wordlist). In this section, we will examine in-user passphrase generation from both types of passphrases and analyze their properties.

3.1 User passphrases

To examine the class of user passphrases, we need to have a dataset compromising user passphrases. Unfortunately, unlike passwords, where data breaches are not uncommon and a lot of which have surfaced publicly [1], there is no public dataset of passphrases available. So, we leverage a simple idea: password leak databases often contain long passwords that could potentially be passphrases without a proper delimiter. For this, we devise a segmentation algorithm to identify passphrases from password leak datasets and construct the first user-chosen, in-use passphrase database. We will release the dataset with the final version of the paper for further research on passphrases

Our approach of generating user passphrases. We start with a compilation of prior data breaches, that surfaced in 2018 by 4iQ security firm [1,45,50]. The leaked dataset contains nearly 1.4 billion email-password pairs. We only consider passwords that are longer than 20 characters, or roughly 4 words (given that the average length of an English word is 4.8 characters [10]), there were 5.7 million such unique passwords. Then we segment passwords using a segmentation algorithm. First, we removed potential hash values [50] using a heuristic, removed passwords with '@' symbol in prefix and '.' in suffix and passwords less than nine English letters in them. We used SymSpell library [29] which is parameterized by a unigram distribution V (created using popular word lists [4, 7, 9]) on these cleaned passwords to find segmentations, or divide them into meaningful words. We only consider a password passphrase if it has 3 valid English words of length greater than or equal to three. We found 68,044 unique passphrases from our segmentation algorithm. These passphrases were used by 72,006 users, 73,088 times in total. Most (98.4%) users used only one passphrase (70,854 users), while 1.6% users (1,152 users) had more than one passphrase. The most frequent passphrase was used by 246 users, whereas 98.4% of passphrases were used by only one user (1,082 passphrases were used by more than one user). We show the top three most used passphrases as well as three randomly sampled passphrases from the ones used by only one user in the top row of Figure 8. We inserted '-' mark to show the segmentation. Finally, for checking the coverage of our method, we take 100 random passwords which are more than 20 characters but discarded by our algorithm. we found 18 passwords that could be considered as passphrases but our algorithm failed to detect them. So our passphrase detection algorithm has a false negative rate of only 18%. By construction, we have zero false-positive rates. We put more details on our method in Appendix A. We will use these User passphrases (we will call it User dataset) to empirically establish the guessability and memorability users achieve when they are choosing the passphrases themselves.

3.2 System-generated passphrases in use

We focus on password managers to understand the in-use system-generated passphrase generation algorithms. We surveyed 13 popular password managers (like LastPass, 1Password, KeePass) [12]. Among these, we found that 3 of them offer to generate passphrases for the user. 1Password [2] and Enpass [5] use diceware, whereas Keepass [6] use 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) [52, 54], which relies on randomly choosing words from a dictionary and combining them to make a passphrase until the required length of passphrases is reached. 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 [54]. One of the most popular approaches that improve upon the memorability of diceware passphrases is the *template based diceware* [8] or TemplateDice.

TemplateDice. This algorithm has a dictionary of English

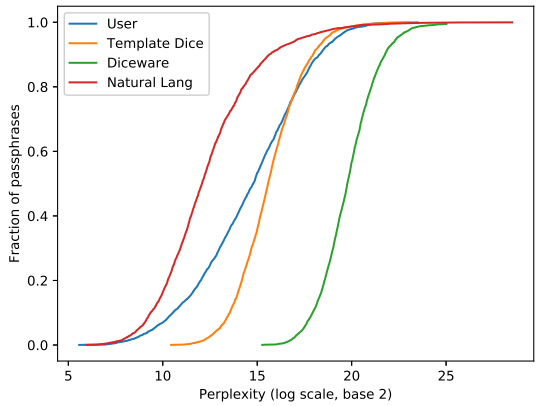

Figure 1: CDF of logarithm of Perplexity to the base 2 for Diceware, TemplateDice and User passphrases, along with natural language phrases for baseline. We observe that User are the closest to the natural language with TemplateDice being almost comparable. The diceware passphrases are much worse in comparison.

words segregated based on various parts of speech tags and has 27 syntactic templates for the English language, whose components are the tags, embedded within the algorithm [13]. The idea of using syntactic templates has handled the issue of memorability very well, as However, this approach has compromised on the security of the passphrases (Section 5.1) and sacrificed the extendability to generate arbitrary length strong passphrases as we will see next while evaluating the property of these passphrases.

3.3 Properties of in-use passphrases

In this section, we first demonstrate one defining characteristic of User passphrase is that it is close to natural language than system-generated ones. Although TemplateDice is somewhat between Diceware and User in terms of being close to natural languages, it shows a plateauing guessrank with increased length—affecting the security.

Linguistic propoerties To verify how close the passphrases generated by the various systems (and user passphrases) are to the natural language, we measure the probability of the passphrases we collected using a natural language model GPT-2 [51]. GPT-2 model is trained to predict the next word given a sequence of words. The model is widely used for natural language text generation [3]. We sample around 3000 passphrases of similar length distribution 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 closer to natural language text. The distribution of the perplexity score [**?**] for both the system (Diceware and TemplateDice) and User passphrases are shown in Figure 1. User passphrases have a very low perplexity value which is similar to the perplexity of a natural language, 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 have a poor performance in this aspect.

Issue with TemplateDice : Although TemplateDice improved upon Diceware, it comes with a compromise in security. On further investigation, we note two major shortcomings for TemplateDice. First, TemplateDice is not scalable as the information required by the system are all internally encoded within the algorithm [13]. Any extension will potentially require a linguist to create new syntax rules and contextual wordlists (assuming it's possible for large passphrases). Second, and more importantly, the security of the passphrases does not scale well with length. We note that the guessranks of these passphrases gets saturated around length 8—guessrank of 8-word passphrase is nearly the same as that as of 13-word. The potential reason is the constraint imposed by the underlying hardcoded and extremely limited syntax rule patterns of TemplateDice. The detailed result is in Appendix B.

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 design a novel constrained Markov Model-based optimization technique in Section 4.5 which is employed to generate passphrases.

4 MASCARA: Optimizing guessability and memorability

To overcome the limitations of User passphrases and system-generated passphrases discussed previously, we design MASCARA: an automated passphrases generation framework that can generate memorable as well as hard to guess passphrases. To do so, MASCARA uses a constrained generative process by modifying a generative Markov model. The constraints are based on approximate memorability and guessability metric that we define. In this section, we present metrics to measure memorability and guessability and incorporate them into a Markov model to design MASCARA.

4.1 Memorability of passphrases

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 have to quantify memorability, and to that extent, we use the notion of character error rate (CER), which is the rate of error per character while

typing the text. Prior works [41, 44, 48] noted that CER is widely accepted as a proxy for memorability.

Leiva et al. however tried to obtain memorable sentences, representative of the corpus, and complete. We only want to ensure generating memorable passphrases, which 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 [41]. We calculated various parameters for each phrase in the dataset like frequency of occurrence, out of vocabulary words, the average frequency of occurrence, etc. With these data, we find the statistically significant correlation of each feature concerning CER and found three statically significantly correlated signals.

Unigram probability (L_1) of a phrase. Calculated as the sum of the log of unigram probabilities of the individual words in the phrase. Phrases with a higher L_1 are likely to be more common, consisting of frequent words, and hence, easier to memorize [36]. 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 supported by its high statistical significance ($p \approx 0$) and high negative correlation ($r = -0.84$) with CER.

Standard deviation (σ_{chr}) of the number of characters per word. Higher variability in the number of characters per word 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. [44], only considered unigrams to distinguish memorable sentences. However, in our work, we also used bigram as identified by our experiment. Higher bigram probability also helps maintain syntactic structure, or the "lexical distinctiveness" to increase memorability [24].

Note that, computation of L_1 and L_2 requires a universal corpus. In this work we use the Wiki-5 dataset (Section 4.3) for the purpose. The model was then fitted according to a generalized linear regression [ref] 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 $\text{CER}(s) = \alpha_1 \cdot L_1(s) + \alpha_2 \cdot L_2(s) + \alpha_3 \cdot \sigma_{\text{chr}}(s)$, with $\alpha_1 = -3.42\text{e}{-2}$, $\alpha_2 = -6.46\text{e}{-3}$ and $\alpha_3 = 1.19\text{e}{-4}$.

4.2 Guessability of passphrases

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 higher the guessrank, lower the guessability. To calculate the guessrank of each passphrase, we simulate any cracking algorithm with the support of suitable training data [17, 39]. The cracking algorithm will then have a probability of generation associated with each model. We employ a Monte-Carlo simulation [26] that uses this probability to calculate the guessrank of the passphrase as discussed below.

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

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

A probability $A[i]$ corresponds to a rank $C[i]$, hence to estimate the guess rank of a passphrase α, we lookup for index j of the rightmost $A[j]$ value in A such that $A[j] > \mathcal{M}(\alpha)$ through binary search. The estimated guessrank is simply $C[j]$. Effectively guessrank depends on the probability assigned to a passphrase.

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 [26]. But research has 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 [59]. 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.

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 which can be used to estimate the guessrank of passphrases and have been trained in suitable training data. Ur et al. [59] 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, we extend the idea to passphrases by taking the minimum 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. Some system-specific models have also been considered to obtain a more accurate approximation of the guessrank, which has been explained in Section 5.

With the framework of guessability and memorability properly defined, we now move on to examine the dataset MASCARA uses for the generation of passphrases.

4.3 Curating a corpus for quantifying memorability and guessability

Our metric for measuring memorability in Section 4.1 requires a universal corpus and the MASCARA needs a bigram Markov model for the generation of passphrases. We use Wikipedia data as a corpus of human-generated text data.

Dataset. We use a recent dump of Wikipedia articles [2] and used the Wikimedia Pageview API client to obtain the top 5% articles based on the page view count, aggregated over the last five years. We then clean the data by removing all tags, URL links, and captions, and used case-folding [22]. We also remove words with less than three characters, as well as numeric or alphanumeric words. Our final data contained 8,210 articles with over 29 million total words and more than 455 thousand unique words. We refer to this dataset as Wiki-5, and use it as a universal corpus for CER estimation throughout the paper and also use this corpus to generate secure and memorable passphrases from MASCARA.

Training bigram Markov model. Next we trained a bigram Markov model on Wiki-5 data. Specifically, we construct all word-bigrams and store them with the number of times they appear in the dataset. We will refer to this model as $\mathcal{M}$, and the log probabilities of unigram and bigrams in the dataset as L_1 and L_2. Thus $L_1(w) = \log(\frac{f_w}{\sum_w f_w})$ and $L_2(w, w') = \log(\frac{f_{ww'}}{\sum_{w''} f_{ww''}})$ and all logarithms are over base 10. Note, we also assume $\mathcal{M}.\mathsf{next}(w)$ as a function that returns all the words that appear after w in the corpus. More details about the wiki data clearning and bigram model is in Appendix C

4.4 Tuning guessability and memorability

We estimate CER and guessrank with the trained Markov model which can thus be used to optimise both memorability and guessability. We use a bigram Markov model trained on the Wiki-5 dataset as one of the models for estimating guessrank for the MASCARA passphrases.

Optimizing memorability. Since we are trying to generate memorable passphrases, we focus on the syntactic structure, as well as the usage of simpler words, to help increase the memorability of the passphrase. We thus introduce the generation probability of a passphrase, based on the various parameters discussed in Section 4.1. As discussed earlier, we express CER for a passphrase as $\text{CER}(s) = \alpha_1 \cdot L_1(s) + \alpha_2 \cdot L_2(s) + \alpha_3 \cdot \sigma_{\text{chr}}(s)$, and to generate memorable passphrases we aim to optimise $\text{CER}(s)$ by controlling the parameters it depends on ($L_1(s), L_2(s)$ and $\sigma_{\text{chr}}(pp)$).

Optimizing guessability. We use the same Wiki-5 corpus to train a bigram Markov model and also to generate the probability distribution of every unigram and bigram the MASCARA uses for generation of passphrases. We then use the Markov model as one of the algorithms used in the estimation of guessrank of the passphrases generated by MASCARA as shown in Section 4.2. We use the unigram and bigram probabilities for calculating the L_1 and L_2 of passphrases for the estimation of CER.

4.5 Generative model

Once we have curated our corpus, and the metrics have been suitably defined, we start generating passphrases.Recall that in our case CER is heavily dependent (and linear combination) of its factors. The equation obtained previously for the CER of a passphrase (using Wiki-5 dataset), helps us have a better estimate to optimize our generated sentences. For ensuring high guessrank too, we try to maintain a trade-off between these two metrics to generate syntactic and secure passphrases using a simple idea: high unigram/bigram probabilities ensure high memorability and low guessrank, so choosing the right words with optimum probability might ensure both memorability and security.

Generation. We start generating passphrases based on the current word. In subsequent words, we evaluate the whole support based on thresholds. For every state change, we recheck our CER and guessrank estimates to obtain a reasonable choice of word.

For the successful generation of passphrases, we pass the trained Markov model to the MASCARA along with the desired length of a generation, L, to generate passphrases adhering to our constraints as discussed below.

We begin with the start token $<\text{s}>$, and the first word appended to the string is from the list of words a sentence begins within the corpus provided that is not a stop word. *Stop words* are a set of commonly used words in any language. Some examples in English - the, are, and, over, etc. We do this to ensure less predictability in our passphrases as there is a high probability of the generated passphrase starting with a stop word otherwise.

We use a score function modeled on the observation that an approximate CER can be computed incrementally. That is, given a partially generated passphrase $s_i = w_1 \dots w_i$, one can compute the intermediate CER value using the following equation.

$$S(w_1 \dots w_i) = \alpha_1 L_1(w_i) + \alpha_2 L_2(w_{i-1}, w_i) + \alpha_3 \sigma_{\text{chr}}(w_1 \dots w_i)$$

[2] https://dumps.wikimedia.org/enwiki/latest/

Similarly, we also know that the guess rank of the generated passphrases is dependent on the bigram probabilities using the bigram probability L_2 when the Markov model trained on the Wiki-5 corpus is used as the cracking algorithm. Thus we use the following constraint while generating passphrases: $S(w_1 \dots w_i) \leq \theta_1$ and $L_2(w_{i-1}, w_i) \leq \theta_2$, where θ_1 and θ_2 are the two parameters of the system.

Constraint thresholds. We introduce θ_1 and θ_2 as thresholds to be satisfied by every word in the support at every step of generation for it to be in contention for the next probable word. Those that fail in any of the two constraints are removed from the support, and then we choose the word weighted based on their conditional bigram probabilities.

These are empirical thresholds set in a way to impose constraints on a relaxed upper bound estimate for the CER score, S, and a relaxed upper bound estimate for the resultant bigram probability, $L_2(w_{i-1}, w_i)$, while also ensuring that the generated passphrases can still retain their syntactic structure and flow.

Since our CER estimate is a linear fit, and the log probability of the entire phrase is the sum of the log probabilities of its constituent unigrams or bigrams, we can take an estimate of the score obtained at every step as a greedy approach to obtaining a sub-optimal solution, rather than having to traverse through every path from the $<$ s $>$ token exhaustively. This approach is beneficial when one tries to use a different corpus or wants to have control over the weight of the factors influencing the CER estimate.

Equation coefficients: $\{\alpha_i\}_{i=1}^{3}$ are the coefficients of the individual parameters we fit with the regression model trained over the Wiki-5 dataset as the universal corpus (Section 4.4). These essentially determine the weight of each factor in the estimation of a phrase's memorability.

The upper bound of the log bigram probability, θ_2, is set to a suitable fraction of the minimum log bigram probability found in the corpus. Similarly, the threshold for the intermediate CER score (S), θ_1, is set to a fraction of the maximum CER.

Relaxing the thresholds a lot will make the quality of generation similar to that of a normal Markov model and tightening them might result in a reduced sample space. Keeping this in mind, the fractions can be tinkered around as per need. Even though they have a common factor, L_2, between each other, the optimization can be considered independent of each other.

Relaxing the thresholds a lot will make the quality of generation similar to that of a normal Markov model. Since the flow of a sentence is based on qualitative inspection, the thresholds depend on the corpus itself, and thus, can vary depending on the user's need. For example, a corpus with a higher percentage of rare bigrams will automatically generate less memorable sentences, thus requiring us to relax the upper bound for CER and vice versa.

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

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

Figure 2: The MASCARA algorithm. The algorithm generates a passphrase of length l given a bigram Markov model $\mathcal{M}$. Here B is a set of stop words, and $<$ s $>$ and $<$ e $>$ are the start and end symbols used in the Markov model. Here $\leftarrow_{L_1} W$ denotes sampling from the support W but according to the probability distribution assigned by unigram probabilities (without log); similarly $\leftarrow_{\$} W$ denote sampling uniformly randomly from the elements in W.

4.6 The final algorithm, MASCARA

We introduce MASCARA, a step-by-step approach to generate passphrases, under constraints of memorability and guessability, while preserving its syntax and meaning in Figure 2. The actual implementation is an optimised version of the one shown, where we use $O(\log n)$ time for the sampling of the next word, where n is the number of choices, as opposed to the $O(n)$ shown in Figure 2, with the help of some pre-processing.

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

Selecting θ_1 and θ_2. We try to ensure that MASCARA favors rarer bigrams (low θ_2) while maintaining a low intermediate CER score (low θ_1). We also note that $S(\cdot)$ function has L_2 in it, so we can bound the value of S given θ_2. That is to say, if $L_2(w_{i-1}, w_i) \leq \theta_2$, then $\theta_1 \geq S(w_1 \dots w_i) \geq \alpha_1 L_1(w_i) + \alpha_2 \theta_2 + \alpha_3 \sigma_{\text{chr}}(w_1 \dots w_i)$, because α_2 is negative. Thus, for a given θ_1, the value of θ_2 can be bounded. σ_{chr} and L_1 is at least 0, then $\theta_1 \geq \alpha_2 \theta_2$, or $0 \geq \theta_2 \geq \frac{\theta_1}{\alpha_2}$.

For our generation method, we chose θ_2 to be at 80%

of the minimum possible value of L_2 (giving the system a leeway of within 20% of the minimum value), which is -17.4. This will ensure that the generated passphrases contain rare bigrams and thereby high guess rank. Setting $\theta_2 = 0.8 \times -17.4$, gives us $\theta_1 \leq 0.5$, as $\alpha_2 = -0.00646$. We use these values for generation in the design of MASCARA.

Our generation is incremental ensuring the invariant of CER (memorability) and guess rank (guessability). An alternative approach would have been generating the whole passphrase of length l, and then checking if the CER and guess rank constraints are met. Intuitively, we can see that the current approach is much more efficient than the alternative and can generate usable passphrases a lot quicker. This is further corroborated by the data shown below.

Execution time. To evaluate the performance of the final MASCARA algorithm, we compare it with multiple variations and baselines. The baselines we take into consideration are Diceware, TemplateDice, as well as a basic Markov model trained on the Wiki-5 dataset. We also examine the variation of MASCARA where all rejections (according to constraints of θ_1 and θ_2) take place after the entire passphrase is generated by a Markov model (MASCARA $_{\text{end}}$). This variation can be understood as a system that can create optimal passphrases for the constraints imposed. For reasonable comparisons, we keep the rest of the system parameters the same for both these versions.

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. Diceware, TemplateDice and Markov run in 7.85 seconds, 0.33 seconds and 0.05 seconds, respectively. In comparison, the MASCARA takes only 0.08 seconds, which is even comparable to a generic Markov model, and much better due to the guarantees offered by the generated passphrases on their memorability and security. Looking further, the execution of the model that can generate the most optimal passphrases for the set constraints, MASCARA $_{\text{end}}$, takes a humongous 810 seconds to complete its execution.

5 Evaluating passphrases

Our goal is to improve upon the passphrases generated by template-based diceware by resolving its shortcomings while still not losing out on the advantages it offers. In other words, we would like to generate passphrases that are easier to remember while still being hard to guess. In this section, we will evaluate the quality of passphrases generated by MASCARA and compare it with passphrases used by users or generated using other methods. Specifically, we will compare amongst the following five sets of passphrases:

Diceware. Diceware was proposed in [52] for generating passphrases containing a sequence of words selected randomly from a vocabulary. We use the wordlist that is commonly used as vocabulary in this system [18]. These passphrases are in general much harder to guess.

Type	Samples
Diceware	1) dreamscape manchuria dervish verbally 2) clay reactive smasher authentic chrome hamster 3) spindle chemicals griminess waviness vintage stammer agenda sulphate
User	1) mind the fold law you should 2) just another happy ending 3) dont cry over spilt milk
Markov	1) leopold arranged for some users include the war 2) during ultraviolet signals beamed 3) their home delivery and morgan suggested that more
TemplateDice	1) when does a bellboy spike an elect but not a sidebar 2) why does Suzy grumble a redeemer 3) how does my overdone one push those violinists after their sounding
Mascara	1) edge bands influenced how far north south 2) stalin however was offered exclusive control those four 3) graham and republic records

Figure 3: Three randomly sampled passphrases from each group of passphrases we consider for evaluation.

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 [13]. 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, except that we don't impose any constraints on the intermediate steps and sample words weighted on their conditional bigram probability.

User. We identified several passphrases used by users from prior password leaks, as described in Section 3. These passphrases are used by users and therefore, should be very easy to remember.

Mascara. Finally, we consider the model we propose — MASCARA — which also internally uses a bigram

Markov model trained on the Wiki-5 dataset, with several control parameters to ensure the generated passphrase has higher memorability while maintaining a high guessrank (Section 4.5 and 4.5 details training and generation of passphrases in MASCARA).

A few random samples of passphrases from each set mentioned above are shown in Figure 3. We compare the memorability of these five sets of passphrases measured by CER and the strength as measured by their guess rank.

Test sample. We generated 1M passphrases from template-based diceware, and following the same distribution of lengths, we generated 1M passphrases from each of Diceware, Markov and MASCARA. We use these as the test sample in the following evaluations. 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 bias. Moreover, the length distribution of these User passphrases is different from those used above, as users often tend to utilise 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.

5.1 Strength of the passphrases

We measure the strength of a passphrase based on their guess rank (q). As discussed in Section 4.2, we use the *min auto* approach by simulating multiple guessing algorithms to estimate the guessrank of a passphrase which is a close approximation of a real-world scenario. We will now elaborate on the different guessing algorithms used.

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. For that scenario, we have 10M system-generated passphrases from Diceware, Markov, Mascara and TemplateDice, on which an attacker can train his cracking algorithm. In the case of User passphrases, the attacker trains on a smaller set of 70k passphrases.

For all of the different sets under evaluation, the attacker trains a corresponding 2-gram, 3-gram word-based probabilistic Markov model and a 4-gram, 5-gram, 6-gram character-based probabilistic Markov model [47]. 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 4.2, we can estimate the guessrank 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 models is then calculated.

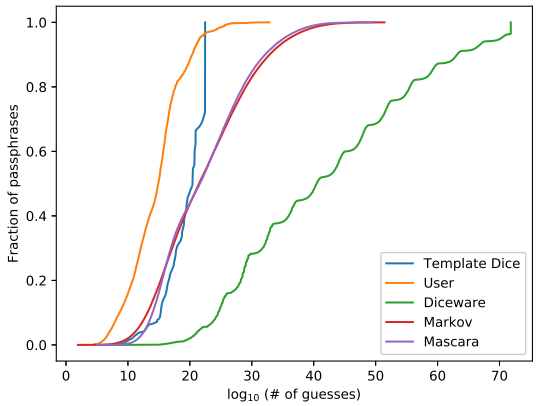

Figure 4: Comparing the strength of different sets of passphrases by evaluating their guessrank. Here, a cumulative distribution frequency of log guessrank (base 10) is shown. We can see that diceware passphrases are the most secure, while the user ones being the most predictable.

Other than the above mentioned probabilistic Markov models (used for all the sets) trained on the passphrases, two other system-specific models are also included during the guessrank estimation:

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. The rest of the guessrank estimation is similar to the process described above.

Template based estimation. An attacker can use the fact that the passphrases generated by TemplateDice are finitely bounded by the templates that are being used. Thus, a guessrank for a passphrase generated by the algorithm can be guessed by trying out all the possible passphrases across all templates. To simulate this process, we first find the number of passphrases each template can produce. We then randomly choose templates one by one until the source template for that passphrase is chosen (we remember the source template so that we can estimate the guessrank, it is not available to the attacker) and count all the passphrases that the attacker would have enumerated by then, which will give us the guessrank for that passphrase. This guessing strategy is exactly the cause for the bounded guessrank for TemplateDice.

Results. We show the (estimated) guess ranks of the passphrases in the test samples in Figure 4. Diceware passphrases are the most secure with 50% of passphrases requiring at least 10^{40} guesses. On the other end of the spec-

trum, 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 [59]. 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. Template-based diceware needs 10^{22} guesses for at least 20% of the passphrases, whereas Mascara significantly improve upon it and require over 10^{30} guesses.

However, we argue that memorability of passphrases is another important criterion that should be considered while picking a passphrase. We discuss the memorability of passphrases next.

5.2 Memorability of passphrases

Passphrases must be memorable while being difficult to guess to be usable in practice. In this section, we measure the memorability of the passphrases in the test samples based on the character error rate (CER) estimate we devised in Section 4.1. CER estimates the probability of making an error while typing from memory (and not the actual #characters that one might get wrong).

Results. The distribution of CER of the passphrases are shown in Figure 5. As expected, Diceware passphrases have a very high CER — 50% passphrases have CER of more than 17% — meaning that users are likely to make a mistake every 6 characters they type, which indicates a very low memorability. 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 between User and Diceware. 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.

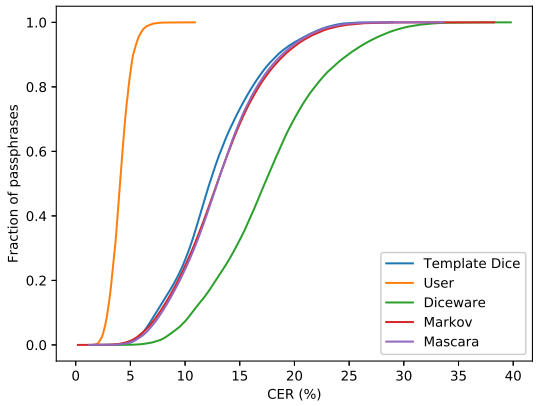

Figure 5: Distribution (CDF) of CER (Character Error Rate) for passphrases among the various samples. Diceware passphrases have the highest CER, making it the least memorable and User passphrases have the lowest CER which in turn means that they are easiest to remember.

Takeaway. We measure the guessrank of the passphrases using the *min auto* approach employing various guessing models to estimate the guessrank as close to what an adversary in a practical scenario might achieve. The memorability of the passphrases is determined by a commonly accepted proxy, CER.

Although we would like to increase the guess rank of passphrases while reducing the CER, they are inversely correlated (intuitively, more structure to passphrases leads to high predictability) and thus one cannot be lowered without increasing the other. We tried to find a sweet spot that allows us to increase as much guessing resistance as possible while keeping the CER values close to what user-chosen passphrases enjoy.

The results discussed above show that MASCARA has overcome all the limitations of TemplateDice discussed in Section 3.3 and is much more effective to use in practical scenarios. 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.

In the next section, we carry out user studies and perform statistical tests on the data obtained to show the memorability variation across the different system generated passphrases.

Model	Part1 count	Part2 count	Return rate
Mascara	72	61	84.72%
TemplateDice	78	63	80.76%
Markov	69	41	59.42%
Diceware	82	50	60.97%

Figure 6: The distribution of the participants who participated in Part 1 as well as the participants who returned for Part 2 across the different models.

6 User Study

We used CER to estimate the difficulty of memorizing various passphrases (in Section 5.2) analytically. However, to understand if indeed finalMASCARA generated passphrases are memorable to the users we ran a user study.

Ethics: As the primary authors' institution did not have an official ethics review board, we did not obtain any official ethics review of the study. However, we discussed extensively with our peers and followed best practices of ethical research (e.g., principles set by Belmont Report [15]). The study asks for no sensitive (e.g., actual in-use passphrases) or personally identifiable information (such as email id). We also removed the worker ids from survey results during analysis, ensuring no privacy risk to the participants.

6.1 Design and setup

Our two-part longitudinal survey-based user study was deployed on crowdsourcing platform Prolific where we assigned Prolific participants at random to one of passphrase generation algorithms out of TemplateDice, Mascara, Markov, Diceware and show them a passphrase to remember. For each algorithm, we randomly generated passphrases while uniforming choosing passphrase length from one of three length ranges ($\leq$ 7, 8-12, >12). We used the range of ≤ 7 as most of User lies in the range. We recruited participants on Prolific with a greater than 99% approval rate for our two-part survey. We selected US Citizens who are fluent in English and are above the age of 18. All the approved participants from part 1 of the survey were invited to carry out part 2. Our study was designed based on prior work on measuring memorability of login credentials [63].

Part 1. In the first part, we asked each participant 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. Then the participant was asked to authenticate with the passphrase to complete the first part. Participants were asked not to write down or copy their chosen passphrase, and to only rely on memory. They were made to practice their chosen passphrase five times after finalizing their choice. Each participant was asked to authenticate twice - at the end of part 1 of the survey, and the beginning of part 2. 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, we make the participants answer demographics and some generic questions.

Part 2. We invited participants who successfully finished part 1 to return and authenticate again after 48 hours from their completion. Prior works [63] used a two-day interval to test recall of passwords, and hence we used the same period by extrapolating the idea to the domain of passphrases as well. At the end of the authentication, we provided a short survey to assess participants perception of the chosen passphrase and how they may want to modify the passphrase. We paid $0.75 to the participants who completed part 1 and $1.00 for the participants who returned and finished their authentication in part 2, the participants took a total of 10 minutes on average to complete both parts. The survey instrument is in Appendix D.

6.2 Demographics and participants info

A total of 310 Prolific users participated in our user study. Among these participants, we detected invalid responses from 9 participants (2.9%). After excluding those, 301 participants successfully chose the passphrases, practised them, and answered all the survey questions properly. 48 hours after this event, we sent an email asking each of them to return for the authentication in the second phase. Out of 301 participants, 217 participants returned, yielding a return rate of 72.1%. The distribution of these participants across the different models and their return rate is also shown in Figure 6.

Of the 301 users, 70% and 24.2% of them reported their gender as female and male, respectively, while the rest either reported it as non-binary or they did not prefer to reveal it. Among the users who participated, the two highest age groups reported were 18-30 (53.16%) and 31-45 (30.56%). Also, 36%, 29% and 17% of the users reported their highest qualifications as Bachelor's Degree, Some college, Master's Degree, respectively. We found no statistically significant differences across the models in their gender, age group, or highest qualification. Only 13% participants reported working in or having education in IT or related fields.

6.3 Passphrase statistics

Word length. To make sure the statistical analysis yields a proper comparison, we would like the distribution of the word lengths of the passphrases across the different models to be similar. The average word length across the passphrases chosen in part 1 among Mascara, TemplateDice, Markov, and Diceware are 9.19, 8.74, 8.62, and 9.06, respec-

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: The percentage of successful recall, mean and median CER (expressed in percentage) during the authentication after 2 days.

tively. Even though the average word lengths are comparable, to be on the safer side, we perform the KW test to see how similar are the underlying distributions. The KW test fails to reject the null hypothesis ($H = 3.59, p = 0.31$), confirming that the underlying distribution is not significantly different. The distribution of the word lengths of the passphrases across returning participants for part 2 among Mascara, TemplateDice, Markov, and Diceware are 9.42, 8.81, 8.85, and 9.16, respectively. Similar to part 1, the KW test fails to reject the null hypothesis ($H = 2.97, p = 0.39$), establishing that the underlying distribution is not significantly different among part 2 participants.

Passphrase strength. We estimate the strength of the passphrases by their guessrank, which have been calculated beforehand using the process mentioned in Section 5.1 corresponding to each model. The mean log guessrank (base 10) across Mascara, TemplateDice, Markov, and Diceware are 14.80, 11.70, 14.46, and 36.49, respectively. Similarly, the median log guessrank (base 10) across Mascara, TemplateDice, Markov, and Diceware are 14.20, 12.51, 12.86, and 36.05, respectively. From this, we can understand that for a length distribution that is not significantly different from each other, the Diceware and TemplateDice passphrases are the most and least secure, respectively.

To compare the underlying distribution of the guessrank among the four system models, we again carry out the KW test. This time, the KW test rejects the null hypothesis ($H = 157.7, p \approx 0$), confirming that the underlying distributions are significantly different. We then perform the Mann-Whitney U test on all possible pairs, and further establish the fact that event the underlying distributions for each pair is statistically significantly different.

6.4 Recall and CER

In our case, recall is successful if users match every character (including spaces) of the passphrase correctly within the five attempts given to them. Also, we calculate the character error rate (CER) for a particular attempt as the edit distance between the attempt and the original passphrase divided by the number of characters in the original passphrase. From this, CER for a user is calculated as the minimum CER across all attempts.

Figure 10 shows the percentage of users who were able to successfully recall for part 1. Mascara, TemplateDice, Markov, and Diceware have recall rates of 59.72%, 51.28%, 63.76%, and 56.07%, respectively for part 1. The recall rates are very close for each of the algorithms. So, in the case of short term memory, passphrase algorithms do not have a significant impact on the memorability of the passphrases. The same is also brought into light by the fact that most participants entered the passphrase correctly in the first try itself, and very few corrected it on the second try.

Figure 7 shows the percentage of users who were able to successfully recall, as well as the mean and median CER for all users during the authentication after 2 days in part 2 of the survey. Mascara and Diceware are close enough - 26.22% and 24.00%, with Mascara performing slightly better. In the 2nd try, 8.82% and 14.11% who did wrong in the first try authenticated successfully for Diceware and Mascara, respectively. Hence, Mascara passphrases have a better gradual memorability. Since we did not restrict participants from copy/paste actions, the second try removes that caveat as well.

Reasons for potential memorability. We also asked participants why the passphrases they chose look memorable to them. Participants considered TemplateDice passphrases to be most grammatically correct (26.92%) and had the best syntactical flow. In both the categories, Mascara is close to TemplateDice, and does better in usage of words that help participants remember the passphrases better in the long run.

7 Conclusion

In this work, we take the first step towards the systematic generation of memorable yet secure passphrases. We presented a novel in-use passphrase data-set and leveraged the linguistic properties to propose MASCARA, a passphrase generation method. In the process, we also created a framework for measuring memorability and guessability of passphrases. Our exploration, aside from creation of MASCARA, provides multiple important insights. First, our ecologically valid in use User passphrases show that users, left to their device choose weak passphrases to optimize for memorability which is in line with earlier work [21] almost a decade back. Second, we demonstrate that optimizing for memorability in passphrases is important, while system-generated passphrases today optimize primarily for security, users prefer passphrases with proper syntax over a random set of words due to memorability. Finally, our work reveals that there is a trade-off between memorability and guessability, and it is possible to balance these two factors for creating more usable system-generated passphrases. Our high-level observation is likely to translate to other non-English languages which is a fertile avenue for future work. Our work opens up the avenue for future exploration of other dimensions of memorability, such as including the user in the loop or allowing minor modifications (e.g., changing a word) to create personally relatable passphrases.

References

[1] 1.4 billion clear text credentials discovered in a single database. https://bit.ly/3r512M7. Accessed: 2021-01-28.

[2] 1password passphrase generation. https://github.com/1Password/spg. Accessed: 2022-01-31.

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

[4] 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.

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

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

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

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

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

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

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

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

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

[14] AL-AMEEN, M. N., WRIGHT, M., AND SCIELZO, S. 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* (2015), pp. 2315–2324.

[15] BEAUCHAMP, T. L. The belmont report. *The Oxford textbook of clinical research ethics* (2008), 149–155.

[16] BONNEAU, J. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy* (2012), pp. 538–552.

[17] BONNEAU, J. Statistical metrics for individual password strength. In *Security Protocols Workshop* (2012).

[18] BONNEAU, J. Eff's new wordlists for random passphrases, Feb 2018.

[19] BONNEAU, J., HERLEY, C., VAN OORSCHOT, P. C., AND STAJANO, F. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy* (2012), IEEE, pp. 553–567.

[20] BONNEAU, J., AND SCHECHTER, S. Towards reliable storage of 56-bit secrets in human memory. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014), pp. 607–623.

[21] BONNEAU, J., AND SHUTOVA, E. Linguistic Properties of Multi-word Passphrases. In *Financial Cryptography and Data Security* (2012), J. Blyth, S. Dietrich, and L. J. Camp, Eds., Lecture Notes in Computer Science, Springer, pp. 1–12.

[22] CHATTERJEE, R., ATHAYLE, A., AKHAWE, D., JUELS, A., AND RISTENPART, T. password typos and how to correct them securely. In *2016 IEEE Symposium on Security and Privacy (SP)* (2016), IEEE, pp. 799–818.

[23] CHATTERJEE, R., WOODAGE, J., PNUELI, Y., CHOWDHURY, A., AND RISTENPART, T. The typtop system: Personalized typo-tolerant password checking. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 329–346.

[24] DANESCU-NICULESCU-MIZIL, C., CHENG, J., KLEINBERG, J., AND LEE, L. You had me at hello: How phrasing affects memorability. In *Proceedings of the ACL* (2012).

[25] DAS, A., BONNEAU, J., CAESAR, M., BORISOV, N., AND WANG, X. The tangled web of password reuse. In *NDSS* (2014), vol. 14, pp. 23–26.

[26] DELL'AMICO, M., AND FILIPPONE, M. Monte carlo strength evaluation: Fast and reliable password checking. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 158–169.

[27] ESKANDARI, S., CLARK, J., BARRERA, D., AND STOBERT, E. A first look at the usability of bitcoin key management. *arXiv preprint arXiv:1802.04351* (2018).

[28] GAO, X., YANG, Y., LIU, C., MITROPOULOS, C., LINDQVIST, J., AND OULASVIRTA, A. Forgetting of passwords: ecological theory and data. In *27th USENIX Security Symposium (USENIX Security 18)* (2018), pp. 221–238.

[29] GARBE, W. Symspell, 2019.

[30] GHAZVININEJAD, M., AND KNIGHT, K. How to memorize a random 60-bit string. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015), pp. 1569–1575.

[31] GRASSI, P. A., GARCIA, M. E., AND FENTON, J. L. Draft nist special publication 800-63-3 digital identity guidelines. *National Institute of Standards and Technology, Los Altos, CA* (2017).

[32] GUO, Y., ZHANG, Z., AND GUO, Y. Optiwords: A new password policy for creating memorable and strong passwords. *Computers & Security 85* (2019), 423–435.

[33] HUTH, A., ORLANDO, M., AND PESANTE, L. Password security, protection, and management. *United States Computer Emergency Readiness Team* (2012).

[34] JAGADEESH, N., AND MARTIN, M. V. Alice in passphraseland: Assessing the memorability of familiar vocabularies for system-assigned passphrases, 2021.

[35] JOUDAKI, Z., THORPE, J., AND MARTIN, M. V. Reinforcing system-assigned passphrases through implicit learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), pp. 1533–1548.

[36] JUST, M. A., AND CARPENTER, P. A. A theory of reading: from eye fixations to comprehension. *Psychological review 87 4* (1980), 329–54.

[37] KEITH, M., SHAO, B., AND STEINBART, P. A behavioral analysis of passphrase design and effectiveness. *Journal of the Association for Information Systems 10* (02 2009), 63–89.

[38] KEITH, M., SHAO, B., AND STEINBART, P. J. The usability of passphrases for authentication: An empirical field study. *International journal of human-computer studies 65*, 1 (2007), 17–28.

[39] KELLEY, P. G., KOMANDURI, S., MAZUREK, M. L., SHAY, R., VIDAS, T. M., BAUER, L., CHRISTIN, N., CRANOR, L. F., AND HERNANDEZ, J. L. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. *2012 IEEE Symposium on Security and Privacy* (2012), 523–537.

[40] KLEIN, D. V. Foiling the cracker: A survey of, and improvements to, password security. In *Proceedings of the 2nd USENIX Security Workshop* (1990), pp. 5–14.

[41] KRISTENSSON, P. O., AND VERTANEN, K. Performance comparisons of phrase sets and presentation styles for text entry evaluations. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces* (2012), pp. 29–32.

[42] KUO, C., ROMANOSKY, S., AND CRANOR, L. F. Human selection of mnemonic phrase-based passwords. In *Proceedings of the second symposium on Usable privacy and security* (2006), pp. 67–78.

[43] LEE, K.-W., AND EWE, H.-T. Passphrase with semantic noises and a proof on its higher information rate. In *2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)* (2007), IEEE, pp. 652–655.

[44] LEIVA, L. A., AND SANCHIS-TRILLES, G. 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* (2014), pp. 1709–1712.

[45] LI, L., PAL, B., ALI, J., SULLIVAN, N., CHATTERJEE, R., AND RISTENPART, T. Protocols for checking compromised credentials. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 1387–1403.

[46] LIU, Y., LI, R., LIU, X., WANG, J., ZHANG, L., TANG, C., AND KANG, H. An efficient method to enhance bitcoin wallet security. In *2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)* (2017), pp. 26–29.

[47] MA, J., YANG, W., LUO, M., AND LI, N. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy* (2014), pp. 689–704.

[48] MACKENZIE, I. S., AND SOUKOREFF, R. W. A character-level error analysis technique for evaluating text entry methods. In *Proceedings of the second Nordic conference on Human-computer interaction* (2002), pp. 243–246.

[49] MELICHER, W., UR, B., SEGRETI, S. M., KOMANDURI, S., BAUER, L., CHRISTIN, N., AND CRANOR, L. F. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 175–191.

[50] PAL, B., DANIEL, T., CHATTERJEE, R., AND RISTENPART, T. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 417–434.

[51] RADFORD, A., WU, J., CHILD, R., LUAN, D., AMODEI, D., AND SUTSKEVER, I. Language models are unsupervised multitask learners. *OpenAI blog 1*, 8 (2019), 9.

[52] REINHOLD, A. The diceware passphrase home page (1995). https://theworld.com/~reinhold/diceware.html.

[53] RÉNYI, A., 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* (1961), The Regents of the University of California.

[54] SHAY, R., KELLEY, P. G., KOMANDURI, S., MAZUREK, M. L., UR, B., VIDAS, T., BAUER, L., CHRISTIN, N., AND CRANOR, L. F. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the eighth symposium on usable privacy and security* (2012), pp. 1–20.

[55] SPECTOR, Y., AND GINZBERG, J. Pass-sentence a new approach to computer code. *Computers Security 13*, 2 (1994), 145–160.

[56] STOBERT, E., AND BIDDLE, R. The password life cycle: user behaviour in managing passwords. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)* (2014), pp. 243–255.

[57] UR, B., ALFIERI, F., AUNG, M., BAUER, L., CHRISTIN, N., COLNAGO, J., CRANOR, L. F., DIXON, H., EMAMI NAEINI, P., HABIB, H., JOHNSON, N., AND MELICHER, W. Design and Evaluation of a Data-Driven Password Meter. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017), CHI '17, Association for Computing Machinery.

[58] UR, B., KELLEY, P. G., KOMANDURI, S., LEE, J., MAASS, M., MAZUREK, M. L., PASSARO, T., SHAY, R., VIDAS, T., BAUER, L., CHRISTIN, N., AND CRANOR, L. F. How does your password measure up? the effect of strength meters on password creation. In *21st USENIX Security Symposium (USENIX Security 12)* (2012), USENIX Association, pp. 65–80.

[59] UR, B., SEGRETI, S. M., BAUER, L., CHRISTIN, N., CRANOR, L. F., KOMANDURI, S., KURILOVA, D., MAZUREK, M. L., MELICHER, W., AND SHAY, R. Measuring Real-World accuracies and biases in modeling password guessability. In *24th USENIX Security Symposium (USENIX Security 15)* (Washington, D.C., Aug. 2015), USENIX Association, pp. 463–481.

[60] VAN ACKER, S., HAUSKNECHT, D., JOOSEN, W., AND SABELFELD, A. 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* (Mar. 2015), CODASPY '15.

[61] WANG, D., ZHANG, Z., WANG, P., YAN, J., AND HUANG, X. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (2016), pp. 1242–1254.

[62] WEIR, M., AGGARWAL, S., COLLINS, M., AND STERN, H. 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* (New York, NY, USA, 2010), CCS '10, Association for Computing Machinery, pp. 162–175.

[63] WOO, S. S. How do we create a fantabulous password? In *Proceedings of The Web Conference 2020* (2020), pp. 1491–1501.

[64] WOO, S. S., AND MIRKOVIC, J. Improving recall and security of passphrases through use of mnemonics. In *Proceedings of the 10th International Conference on Passwords (Passwords)* (2016).

[65] YAN, J., BLACKWELL, A., ANDERSON, R., AND GRANT, A. Password memorability and security: empirical results. *IEEE Security Privacy 2*, 5 (2004), 25–31.

[66] ZVIRAN, M., AND HAGA, W. J. A comparison of password techniques for multilevel authentication mechanisms. *Comput. J. 36* (1993), 227–237.

A Detecting User-generated passphrases

A.1 User passphrases

To examine the class of user passphrases, we need to have a dataset compromising user passphrases. Unfortunately, unlike passwords, where data breaches are not uncommon and a lot of which have surfaced publicly [1], there is no public dataset of passphrases available. But we notice that several password leak databases contain long passwords that could potentially be passphrases without a proper delimiter. For this, we devise a segmentation algorithm to identify passphrases from password leak datasets and construct the first user-chosen, in-use passphrase database. We describe in this section how we extracted the passphrases and will release the dataset with the final version of the paper for further research on passphrases

Password leak dataset. We use a compilation of prior data breaches, that surfaced in 2018 by 4iQ security firm [1]. The leaked dataset contains nearly 1.4 billion email-password

pairs. Prior research on passwords has used this leak [45,50]. We use this dataset to extract potential in-use passphrases.

To find passphrases in this dataset, we only consider passwords that are longer than 20 characters, or roughly 4 words (given that the average length of an English word is 4.8 characters [10]). We found 5.7 million unique passwords that are longer than 20 characters. These passwords were used by 5.9 million users (identified by email addresses in the dataset) [3] 6.2 million times in total.

Segmenting passwords. As these passwords are selected from a compilation of password leaks, many of them are potentially hash values [50]. We remove these hash values using a heuristic-based identification algorithm. We check if the password only contains hexadecimal characters and if so, we flag them as hash values and remove them. This removed 1.9 million unique passwords. We also find many of the 20-or-more character passwords look like an email addresses or have some parsing errors. We removed such 1.5 million passwords that contain an '@' symbol with a prefix and has '.' in its suffix. We also removed 0.4 million passwords that had less than nine English letters in them, as that is the minimum number of characters necessary to form a three-word passphrase, with each word at least three characters long (See details below). This left us with 1.8 million passwords that we then test using our passphrase segmentation algorithm.

We used a standard NLP task of doing word segmentation of noisy text for segmenting passphrases. Specifically, we used a SymSpell library [29] which is parameterized by a unigram distribution V. Given a password s, it can segment to create a sequence of words $(w_1, w_2, \ldots, w_l)$, such that $\prod_{i=1}^{l} \Pr[w_i]$ is maximized. SymSpell also tolerates a specified amount of variations of the words in V. For segmenting passphrases, we used the common English dictionary from NLTK [4], names of countries and popular cities [7], and common first names in the US [9]. We set the frequencies to be 1 for all words. We added password specific edits, such as L33t speak transformations to account for modification of words in a password. We only consider a password passphrase if it has 3 valid English words of length greater than or equal to 3.

Resulting passphrases dataset. Using our segmentation approach, we found 68,044 unique passphrases from our segmentation algorithm. These passphrases were used by 72,006 users, 73,088 times in total. Most (98.4%) users used only one passphrase (70,854 users), while 1.6% users (1,152 users) had more than one passphrases. The most frequent passphrase was used by 246 users, whereas 98.4% of passphrases were used by only one user (1,082 passphrases were used by more than one user). We show the top three most used passphrases as well as three randomly sampled passphrases from the ones used by only one user in the top row of Figure 8. We inserted '-' mark to show the segmentation.

[3] We combined all the passwords (of any length) belonging to the same email. We further combined the emails (and the corresponding passwords) if the two emails share the same username (the part before the '@' symbol) and if they have a common password. We ignored the users with more than 1,000 passwords, as they are unlikely to be real user accounts. Such preprocessing was also done in [50].

Type	Examples

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

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

Ineligible	`speedtriple123456789`
	`21101975-invalidlogin`
	`newjob2thomapink_socks08`

Common names	`KatherineCarrasquillo`
	`zuleimahernandez1230`
	`dobrovolskayatatiana`

Non-phrasal	`ltdjxrfgtctybz27102003`
	`oilgurtalococsecnarf`
	`903kingdalonsbfreitag`

Figure 8: Figure shows different types of passwords that we analyzed. In the top row, we show the most common passphrases, as well as random samples that we were able to extract from passwords, while in the bottom three rows, we show the passwords that we do not consider as passphrase, and the categories they fall into.

We also look into the passwords that have not been considered as passphrases to gauge the false negative rates of our algorithm. We take a random sample of hundred passwords from the 1.8 million passwords that could be potentially passphrases. We found primarily three reasons for passwords being discarded: (a) *Ineligible* (passwords having less than 3 words), (b) *Common names* (passwords that are common names, and our segmentation algorithm failed to segment them meaningfully), and (c) *Non-phrasal* (passphrases that are just a mix of letters, digits, and symbols that do not segment into any meaningful sequence of words.) We show samples of these three types of passwords not detected by our algorithm at the bottom three groups of rows in Figure 8.

B Shortcoming of TemplateDice

We show the guessrank of an adversary who does brute force guess of the syntax rules and then wordlists on TemplateDice passphrases in Figure 9. We note that the guessranks of these passphrases gets saturated around length 8—guessrank of 8-word passphrase is nearly the same as that as of 13-word. The potential reason is the constraint imposed by the underlying hardcoded and extremely limited syntax rule patterns

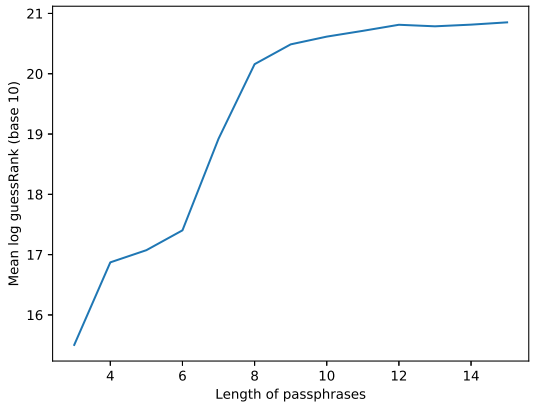

Figure 9: Variation of mean $\log_{10}$ guessrank across length for the TemplateDice passphrases. We observe that guessranks reached a plateau after length eight.

of TemplateDice

C Creating wiki dataset

Our metric for measuring memorability in Section 4.1 requires a universal corpus and the MASCARA needs a bigram Markov model for the generation of passphrases. For that, we create a user-generated text corpus from Wikipedia.

Dataset. We will use Wikipedia data as a corpus of human-generated text data. We argue 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. We use a recent dump of Wikipedia articles [4] to extract content based on the titles of the articles. We use the `pageviewapi` [5], which is the Wikimedia Pageview API client, to obtain the top 5% articles based on the page view count, aggregated over the last five years. We set the granularity of the views as monthly. This only strengthens the scope of our corpus, without having to use a huge dataset, to cover all possible domains. We then use the MediaWiki API, to extract content based on titles.

The raw Wikipedia text contains tags and URLs. As they are not representative of human-generated text and are likely to interfere with our estimation and evaluation methodologies, we pre-process our text dataset by removing all tags, URL links, and captions. We used regular expressions, such as checking for a URL link with or without 'HTTP' or 'HTTPS' followed by at least three alphanumeric chunks separated by '.' and valid extensions from the third chunk [6] for finding and removing them. We also remove uncommon punctuation symbols, such as ';;', '__', etc. We normalized all words to lower case as we will only consider lower case words in our passphrase (such passphrases are also easy to type [22]). We also remove words with less than three characters, as well as numeric or alphanumeric words, to keep the generation consistent with User passphrases. Finally, we obtained a textual corpus containing 3.3 million sentences (delimited by a period '.', from 8,210 Wikipedia articles, containing a total of 2,95,02,034 words (and 4,55,614 unique words).

We refer to this dataset as Wiki-5, use it as a universal corpus for CER estimation throughout the paper and also use this corpus to generate secure and memorable passphrases from MASCARA. First, we explain the training of a bigram Markov model with this corpus.

Training bigram Markov model. We first train a bigram Markov model on Wiki-5 data. To do so, we add a beginning of sentence $<$ s $>$ and end of sentence $<$ e $>$ marker to each sentence we identified in the dataset. Then we construct all bigrams and store them in a bi-level dictionary counting the number of times they appear in the dataset. Words are considered a contiguous sequence of English letters delimited by non-alphabetic characters. We did not do any stemming or lemmatization of the words found.

We will refer to this model as $\mathcal{M}$, and the log probabilities of unigram and bigrams in the dataset as L_1 and L_2. Thus $L_1(w) = \log(\frac{f_w}{\sum_w f_w})$ and $L_2(w, w') = \log(\frac{f_{ww'}}{\sum_{w''} f_{ww''}})$ and all logarithms are over base 10. Note, we also assume $\mathcal{M}.\mathsf{next}(w)$ as a function that returns all the words that appear after w in the corpus.

D User Study Questions

D.1 Part 1

1. Do you use passphrases as a secret for logging in to any of your online accounts?

- ○ Yes
- ○ No

Why? (1 to 2 sentences) _____

2. How did you generate your passphrase(s)? Choose all that apply.

- ○ Used an online tool [also write names, if you remember]: _____
- ○ Self-generated - using a quote from a poem, movie, or book
- ○ Self-generated - using a combination of random words

[4] https://dumps.wikimedia.org/enwiki/latest/
[5] https://pypi.org/project/pageviewapi/
[6] https://www.regextester.com/93652

○ Other: _____

3. For each of the questions below please choose the option which applies most for you

- Do you write down your login credentials to remember them?
  Never ○ ○ ○ ○ ○ Always
- How often do you use the same login credential for multiple websites?
  Never ○ ○ ○ ○ ○ Always

4. Why did you choose this particular passphrase among the ones shown? (1 - 2 sentences): _____

5. Please select options from below which have positively affected your memorability of the chosen passphrase

○ Contains frequently used words.

○ Grammatically correct

○ Fewer words to remember as it contains common words

○ Flows like an English phrase

○ Other: _____

6. Which age group do you belong to?

○ 18-30

○ 31-45

○ 46-60

○ 60+

7. Which gender do you identify yourself most with?

○ Male

○ Female

○ Non-Binary / Third Gender

○ Prefer not to say

8. What is the highest degree or level of school you have completed?

○ Some high school

○ High school

○ Some college

○ Trade, technical, or vocational training

○ Associates degree

○ Bachelors degree

○ Masters degree

○ Professional degree

○ Doctorate

○ Prefer not to say

9. Which of the following best describes your educational background or job field?

○ I have an education in, or work in, the field of computer science, engineering, or IT.

○ I do not have an education in, or work in, the field of computer science, engineering, or IT.

○ Prefer not to say.

D.2 Part 2

1. What is your preferred length for a passphrase you would want to use (the number of words)?

○ 7 or less words

○ 8-12 words

○ >12 words

2. Please briefly explain your choice of preferred length for passphrases (1-2 sentences): _____

3. If you have to use a passphrase, then while choosing the passphrase, would you prefer to prioritize security or memorability?
Prioritize only Memorability ○ ○ ○ ○ ○ Prioritize only Security

4. Currently, do you feel changing your chosen passphrase slightly (e.g., a few characters) would have made it more memorable for you without making it easy to guess for others?

○ No, I dont want to change the passphrase.

○ Yes, I want to change my chosen passphrase to _____

Why? _____

5. After participating in this study, how likely are you to use passphrases as your login credential for some online accounts instead of passwords?
Not Likely At All ○ ○ ○ ○ ○ Very Likely
Why? _____

E User Study Results

Model	Recall	Try1 recall	Try2 recall
Mascara	59.72%	50.00%	56.94%
TemplateDice	51.28%	43.59%	47.44%
Markov	63.77%	53.62%	57.97%
Diceware	56.10%	42.68%	51.22%

Figure 10: The percentage of successful total recall, successful recall by try 1 and successful recall by try 2 during the authentication in part 1 among all participants

Model	Recall	Try1 recall	Try2 recall
Mascara	46.15%	38.46%	46.15%
TemplateDice	28.57%	21.43%	28.57%
Markov	14.29%	14.29%	14.29%
Diceware	23.08%	7.69%	15.38%

Figure 11: The percentage of successful total recall, successful recall by try 1 and successful recall by try 2 during the authentication in part 2 among the users with passphrases of length less than or equal to 7

Model	Mascara	TemplateDice	Markov	Diceware
Grammatically Correct	20.83%	26.92%	17.39%	7.32%
Flows like an English phrase	52.78%	60.26%	47.83%	28.05%
Fewer words to remember	41.67%	39.74%	57.97%	39.02%
Contains frequently used words	47.22%	24.36%	43.48%	41.46%

Figure 12: The potential factors for memorability that were asked of users during the first part of the survey (Q5 of Appendix D.1)