MASCARA: **Systematically Generating Memorable And Secure Passphrases**

Avirup Mukherjee
IIT Kharagpur
avimukh.250696@iitkgp.ac.in

Kousshik Murali
IIT Kharagpur
kousshikraj.raj@gmail.com

Shivam Kumar Jha
IIT Kharagpur
shivam.cs.iit.kgp@gmail.com

Niloy Ganguly
IIT Kharagpur, L3S Hannover
niloy@cse.iitkgp.ac.in

Rahul Chatterjee
University of Wisconsin–Madison
rahul.chatterjee@wisc.edu

Mainack Mondal
IIT Kharagpur
mainack@cse.iitkgp.ac.in

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, while state-of-the-art machine-generated passphrases are difficult to remember.

In this work, we aim to tackle the drawbacks of the 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. We identify and characterize 72, 999 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-generated passphrases are harder to guess than in-use user-generated passphrases, while being easier to remember compared to state-of-the-art machine-generated passphrases. We conduct a two-part user study with crowdsourcing platform Prolific to demonstrate that users have highest memory-recall (and lowest error rate) while using MASCARA passphrases. Moreover, for passphrases of length desired by the users, the recall rate is 60-100% higher for MASCARA-generated passphrases compared to current system-generated ones.

CCS CONCEPTS

• **Security and privacy** → **Usability in security and privacy**.

KEYWORDS

passphrases, authentication, memorability, guessability, dataset

ACM Reference Format:
Avirup Mukherjee, Kousshik Murali, Shivam Kumar Jha, Niloy Ganguly, Rahul Chatterjee, and Mainack Mondal. 2023. MASCARA: Systematically Generating Memorable And Secure Passphrases. In *ACM ASIA Conference on Computer and Communications Security (ASIA CCS '23), July 10–14, 2023, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3579856.3582839

1 INTRODUCTION

Passwords are by far the most popular method for authentication, despite their several limitations: users have to remember many (unique) passwords and, in turn, they tend to choose weak, easy-to-guess passwords. To improve security of user passwords, several alternative tools and strategies are proposed, such as password managers and two-factor authentication mechanisms. However, to secure password managers (and even while two-factor authentication is used), users are required to create and remember a secure and memorable master secret. To that end, *passphrases* are considered an alternative approach to generate memorable yet strong authentication secrets.

Passphrases, unlike passwords, are sequences of words, for example "correct horse battery staple". They are frequently recommended for particularly sensitive scenarios—as the master secret for password managers [30], for locking users' Cryptocurrency wallets (e.g., brainwallet [20]), or protecting SSH keys [49]. Passphrases could also be used to create more memorable passwords. For example, mnemonics created from passphrases can assist in memorizing complex passwords [66]. Thus Passphrases are a secure and user-friendly alternative to passwords [14], while we (slowly) transition toward a "password-less" future of authentication.

Passphrases can be user-generated or system-generated. User-generated passphrases are memorable, often because of their sentence/grammatical structure being aligned with natural text [35, 67] To that end, system-generated passphrases are proposed—such passphrases are difficult for an attacker to guess [8], but are also difficult to remember [31, 57].

Existing approaches to computer-generated passphrases include (a) *Diceware* [54], which picks random words from a given wordlist, and (b) *Template-based Diceware* [25], which generates passphrases that adhere to a small set of pre-selected syntax template (e.g., user-adverb-verb-noun). Shay et al. [57] showed that Diceware passphrases are as difficult to remember as randomly generated passwords. Further, our analysis reveals (Section 3.2) that Template-based Diceware also suffers two major drawbacks: (1) there is a fixed maximum bound on the strength of the generated passphrases, because it uses a handful of specific syntax templates, and (2) there is no systematic way to extend the space of passphrases with more templates (Section 3.3), necessitating a thorough re-design of the approach. Another prior work took an alternative approach—they tried to use techniques like implicit learning *post-passphrase generation* to improve memorability [31].

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 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 note that the current passphrase generation methods like Diceware implicitly trade off memorability for security. In this work, we explore this trade-off and design, build, and evaluate MASCARA which attempts to provide a balanced trade-off between security and memorability. We design MASCARA using insights from a novel in-use English passphrase dataset to ensure good guessability-memorability trade-offs. The key contributions of this work are:

- We used heuristics to identify 72,999 user-chosen English passphrases from prior password leaks. To the best of our knowledge, our dataset is the largest user-chosen passphrase dataset to date.[1] Our algorithm leverages word segmentation in the noisy text to identify these passphrases. The syntactic structures of these passphrases are distinctly different from Diceware—favoring memorability over guessability.
- We created a memorability-guessability measurement framework for passphrases. Building on prior works on the memorability of natural language phrases we identified distinct and important features of a sentence which affects the memorability of a passphrase. We additionally created a Monte-Carlo estimate of the passphrase guess ranks to measure the guessability of passphrases. 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. For passphrases of length 7 or less (preferred by most users) MASCARA provides 1.6x–2x better recall rate than deployed systems like Diceware while maintaining a less than 10% character error rate.

[1]The dataset and code for this work can be found at https://github.com/Mainack/MASCARA-passphrase-code-data.

Limitations: Our study has three key limitations. First, we are not providing a one-step solution to the quest of memorable passphrases. Rather MASCARA takes a principled and complementary approach to enhance today's system-generated passphrases by balancing memorability and security. Second, as with any user study, ecological validity is hard to ensure objectively. In line with earlier work, we made a conscious and earnest effort in our experiments to not nudge our participants to choose or better remember passphrases from any specific system (to preserve the sanctity of recall values) [18]. Participants did not know passphrases shown to them came from which system—our quantitative and qualitative analysis also does not indicate any bias. In fact, if the users believed that our study required to remember their chosen passphrases correctly and thus always choose the one which was easiest to remember and go out of their way to memorize (e.g., using pen and paper), we might not have seen the wide variance in recall values (see Figure 6). Finally, in line with prior work, we only consider English passphrases—it enables comparison with state-of-the-art [7, 9, 31]. Exploring passphrases for other languages is part of our future work.

2 BACKGROUND AND RELATED WORK

2.1 System-generated Passphrases

Passwords are used for authenticating to critical as well as non-critical infrastructures [8]. Unfortunately, several prior works [16] have shown that users tend to choose predictable passwords and reuse them across multiple accounts. To that end, in the past decade, passphrases have been 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* [26]. Intuitively, passphrases are likely to be easier to remember than passwords (due to their closeness to natural language) for users as well as harder to guess (due to their length) for adversaries. Shay et al. [57] demonstrated (using a user study) that even simply using passphrases consisting 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. Today passphrases are used in password managers, cryptocurrency wallets [44], and for securing ssh keys [49].

Several prior works focused on generating and remembering secure passwords (and passphrases) using techniques like contextual cues, portmanteau, or mnemonic based generation [2, 32, 39, 58, 65]. These techniques aim to associate a context (like an image) with the secret. Consequently, our work on generating secure and memorable passphrases is complementary to such techniques of remembering secrets—they can be directly used to improve memorability of MASCARA-generated passphrases.

Prior works however have not systematically investigated the guessability-memorability trade-off. In this work, we formalize the guessability-memorability problem by creating a data-driven framework and leverage the framework to design MASCARA that can generate secure and memorable passphrases.

2.2 Security and threat model

Measuring security of passwords is a well-studied topic [37]. Earlier works considered different approaches the attacker could employ and thus used Markov models, probabilistic CFG, neural networks, etc. for the guessability estimations [47, 50, 59, 61, 62]. However, there is relatively less work in the domain of passphrases.

Using guess rank 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 [40]. However, if users are given control, they generally tend to opt for common phrases, reducing security drastically [39]. In all of these works, the security of passphrases is generally measured through either entropy or user surveys [55].

But later, researchers have shown that *guess rank* is a much more acceptable measure than entropy for measuring the security of a password [60, 64]. The guess rank of a password can be understood as the number of guesses an adversary needs to arrive at the correct password. So higher the guess rank, lower the guessability. Building upon this and earlier works on password meters [47, 59, 61], we use the guess rank metric for measuring the guessability of passphrases in our setup. Specifically, we estimate a guess rank for each passphrase in our setup— the higher the guess rank, 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. We assume the attacker is fully aware of the passphrase generation algorithm and the dataset used in all training. The attacker can generate (offline) as many passphrases as they can (given their computational resources) for any algorithm. The goal of the attacker is to guess the password randomly generated using an algorithm given a large guessing budget (e.g., 10^{15}). Our offline attack [8] model is stronger than an online attack setting where the attacker is limited by the number of guesses they can make. The primary challenge for the attacker is to generate an ordered list of passphrase guesses $w_1, w_2, ..., w_n$ to reach the target passphrase w as early as possible (least number of guesses). The higher the guess rank 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. [60]. First, we used multiple password 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 [6, 36]. For each algorithm, the *guess rank* will be the number of incorrect guesses the particular algorithm used to arrive at the correct passphrase. Since the average guess rank for even a passphrase of medium strength is of the order of 10^{15} (Section 5.1), running the algorithms to find the guess rank is infeasible—we therefore used Monte Carlo simulations to estimate the guess rank (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 guess rank. Ur et al. demonstrated that taking minimum guess rank of all automated approaches (called *min auto*) is a reasonable approximation of real-world cracking scenario [60].

2.3 Measuring memorability of passphrases

Earlier works tried to correlate password memorability with the frequency of passwords, login durations, and even keyboard pattern [21, 27]. Other works used methods like encoding random n-bit strings for generating memorable passwords or using chunks [24]. All of these cases measured memorability of passwords based on user surveys instead of an automated linguistic metric [13, 57, 65]. 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. [15] 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 [38, 46]. 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 two-part 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

Passphrases can be generated by users or by computers (*system-generated*). System-generated passphrases, uses an algorithm and generates pseudorandom passphrases based on a training corpus or wordlist. In this section, we will examine different in-use passphrase 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, resulting in public dataets [11], 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. Thus we devise a segmentation algorithm to identify passphrases from password leak datasets and construct the first user-chosen, in-use passphrase dataset.

Identifying user passphrase from leaked password data. We leverage a compilation of prior password data breaches that surfaced in 2018 by 4iQ security firm [11, 43, 50]. The leaked dataset contains nearly 1.4 billion email-password pairs and has been used in prior works [43, 50, 56].

For our study, we only consider passwords that contain 20 or more ASCII characters (which is roughly 4 words given that the average length of an English word is 4.8 characters [52]). There were 5.7 million such unique passwords. Then, we filtered this list by removing passwords that are potentially hash values [50] — containing only hexadecimal characters or follows popular hash formats [28] — or emails — containing '@' symbol in prefix and '.' in suffix and passwords less than nine English letters in them. Then we segment passwords using segmentation algorithms. Segmentation algorithm tries to split the passwords into a sequence of meaningful words. We tried a hybrid segmention approach (combining two segmentation approaches) for segmenting passwords and effectively detecting user-generated passphrases (details in Appendix A.1).

We found 72, 999 unique passphrases using our algorithm. The most frequent passphrase was used by 258 users, whereas 99.5% of passphrases were used by only six or less users. 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 7. We inserted '-' marks 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 might miss out on passphrases that are heavily modified; but by construction, we have zero false-positives. We put more details on our method in Appendix A. We will use these user generated passphrases (we will call it User dataset) to empirically establish the guessability and memorability users achieve when they are choosing the passphrases themselves. We compared this passphrase dataset with leaked real-world password dataset to check for ecological validity. We found that linguistic properties of our dataset are inline with previous work on passphrases and frequency distribution of the passphrase dataset mirrors that of the original breached password dataset, hinting at ecological validity of our passphrase dataset. Detailed results are in Appendix A.2.

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, such as LastPass, 1Password, KeePass [51]. Among these, we found that three password managers provide a passphrases generation functionality: 1Password [1] and Enpass [19] use Diceware, whereas Keepass [34] uses a template-based version of Diceware (which we call TemplateDice) as their internal algorithm to generate passphrases for users. We describe Diceware and TemplateDice below.

Diceware. The most commonly used system passphrase generation algorithm is Diceware (called Diceware from now on) [54, 57], 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, users find it very hard to remember the passphrases, i.e., the memorability of the passphrases is very low [57]. An in-use approach to improve upon the memorability of Diceware passphrases is the *template based Diceware* [25] or TemplateDice.

TemplateDice. This algorithm has a dictionary of English words segregated based on various parts of speech tags and has 27 syntactic templates for the English language, whose components are the tags, embedded within the algorithm [25]. The idea of using syntactic templates has handled the issue of memorability very well. However, this approach has compromised on the security of the passphrases (Section 5.1) and sacrificed the extensibility 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 closer to natural language than system-generated ones. TemplateDice is somewhat between Diceware and User in terms of being close to natural languages.

Linguistic properties. We randomly sampled 3000 passphrases of similar length distribution from each system (dataset for User) and compute their perplexity [10] using GPT-2 model [53]. Lower perplexity indicates closer to natural language text. We noted that User passphrases have a very low perplexity value similar to a natural language corpus, signifying a key reason for their memorability. On the other hand, the passphrases from TemplateDice is a close second in their resemblance to natural language, almost comparable to user passphrases, which indicates an improvement over the passphrases generated by Diceware that performed poorly in this aspect. Details are in Appendix B.

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 [25]. 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 guess ranks of these passphrases gets saturated around length 8—guess rank 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 (Appendix C).

With these insights, we aim to improve the state of the art by balancing security and memorability of the passphrases. To that end, we design a novel constrained Markov Model-based optimization technique in Section 4.5 for generating passphrases.

4 MASCARA: OPTIMIZING GUESSABILITY AND MEMORABILITY

To overcome the limitations of User passphrases and system-generated passphrases, 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 metrics that we define.

4.1 Memorability of passphrases

In order to improve memorability of system-generated passphrases, we first have to quantify memorability, and to that end, we use the notion of character error rate (CER), which is the rate of error per character while typing the text. Prior works [38, 42, 46] noted that CER is widely accepted as a proxy for memorability.

Leiva et al. [42], however, tried to obtain memorable sentences which are representative of a corpus and are complete. On the contrary, we want to ensure generating memorable passphrases which might not be complete and might not be representative of all passphrases users can memorize. Thus, the signals for memorability we want can be very different.

In order to investigate, 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 [38]. 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 (using pearson product moment correlation coefficient [41]) 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 [33]. In fact, 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 ($p < 10^{-4}$) and positively correlated concerning CER ($r = 0.25$).

Leiva et. al. [42], only considered unigrams to distinguish memorable sentences. However, in our work, we included bigram too as identified by our experiment. Higher bigram probability helps maintain syntactic structure, or the "lexical distinctiveness" to increase memorability [15].

Note that, computation of L_1 and L_2 requires a large 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 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 $\alpha_1 = -3.42 \times 10^{-2}$, $\alpha_2 = -6.46 \times 10^{-3}$ and $\alpha_3 = 1.19 \times 10^{-4}$.

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 guess rank, which is, in fact, the estimated number of tries for an adversary to arrive at the correct passphrase. Recall that higher the guess rank, lower the guessability. To calculate the guess rank of each passphrase, we simulate any cracking algorithm with the support of suitable training data [6, 36]. The cracking algorithm will then have a probability of generation associated with each model. We employ a Monte-Carlo simulation [17] that uses this probability to calculate the guess rank of the passphrase as discussed below.

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

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

A probability $A[i]$ corresponds to a rank $C[i]$, thus we estimate guess rank of a passphrase α with the largest j such that $A[j] > \mathcal{M}(\alpha)$ through binary search. The guess rank is estimated by taking an weighted average of the values of $C[j-1]$ and $C[j]$.

The above process allows us to calculate the guess rank 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 [17]. 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 [60]. This idea can be extrapolated to the domain of passphrases and thus using any single model for estimating the guess rank 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 guess rank of passphrases and have been trained in suitable training data. Ur et al. [60] have shown that for each password taking the minimum of all guess ranks estimated by the various models is a reasonable approximation to the actual guess rank 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 guess rank (Section 5). Leveraging this guessability and memorability framework we next examine the dataset MASCARA uses for passphrase-generation.

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 used 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 [12]. 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.

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

Training bigram Markov model. Next we trained a bigram Markov model on Wiki-5 data. We record all word-bigrams and their frequency of appearance 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 $\mathcal{M}.\text{next}(w)$ as a function that returns all the words that appear after w in the corpus.

4.4 Tuning guessability and memorability

We estimate CER and guess rank using a bigram Markov model trained on the Wiki-5 dataset in this work for estimating guess rank 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. 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 optimize $\text{CER}(s)$ by controlling the parameters it depends on $L_1(s), L_2(s)$ and $\sigma_{\text{chr}}(s)$.

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 guess rank 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 guess rank 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 guess rank, 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 guess rank 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

```
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 $i \le l$ do
    $W' \leftarrow \mathcal{M}.\text{next}(w_{i-1})$
    if $i = l$ then
        $W' \leftarrow W' \setminus B$  /* Remove stopwords */
    /* CER and guess rank constraint */
    $W' \leftarrow \{w \in W' \mid S(w_1 \ldots w) \le \theta_1 \text{ and } L_2(w_{i-1}, w) \le \theta_2\}$
    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 $w_i = \bot$ then
        $w_1 \leftarrow$ GetFirstWord($\mathcal{M}$)  /* No passphrase found; restart */
        $i \leftarrow 1$
    $i \leftarrow i+1$
return $w_1 \ldots w_l$
```

Figure 1: 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 $< \text{s} >$ and $< \text{e} >$ 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.

- 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 $s_i = w_1 \ldots w_i$, one can compute the intermediate CER value using the following equation.

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

Similarly, we also know that the guess rank of the generated passphrases is dependent on the bigram probabilities 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: $S(w_1 \ldots w_i) \le \theta_1$ and $L_2(w_{i-1}, w_i) \le \theta_2$, where θ_1 and θ_2 are two system parameters.

Relaxing thresholds on $thetaone$ and $thetatwo$ 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. 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. We provide a detailed discussion on trade-off and bound of thresholds on θ_1 and θ_2 in Appendix D.

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 1. The actual implementation is an optimized 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 1, 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, 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 $L_2(w_{i-1}, w_i) \le \theta_2$, then $\theta_1 \ge S(w_1 \dots w_i) \ge \alpha_1 L_1(w_i) + \alpha_2\theta_2 + \alpha_3\sigma_{\text{chr}}(w_1 \dots w_i)$, 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 $\theta_1 \ge \alpha_2\theta_2$, or $0 \ge \theta_2 \ge \frac{\theta_1}{\alpha_2}$.

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 $\theta_1 \le 0.5$, 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 demonstrated 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 checked 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, 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 very significantly larger 810 seconds to complete.

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—we will compare the following five sets of passphrases:

Diceware. Diceware was proposed earlier for generating passphrases by random selecting a sequence of words from a vocabulary [54]. We use a wordlist commonly used as vocabulary Diceware [7]. These passphrases are in general much harder to guess (e.g., "clay reactive smasher authentic chrome hamster").

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 [25]. The passphrases generated in such a way are relatively easier to remember (e.g., "when does a bellboy spike an elect but not a sidebar").

Markov. We also use a bigram Markov model trained on the Wiki-5 dataset as a baseline for comparison considering that MASCARA is an enhanced version of the former. The process of passphrase generation is similar to MASCARA. However, we don't impose any constraints on the intermediate steps and sample words weighted on their conditional bigram probability (e.g., "leopold arranged for some users include the war").

User. We identified several user-created passphrases from prior password leaks (in Section 3). These passphrases are user-created, close to natural text and therefore, should be very easy to remember (e.g., "just another happy ending").

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 (e.g., "edge bands influenced how far north south"). Later we detail training and passphrase-generation of MASCARA (Section 4.5 and D).

We add a few random samples of passphrases from each set in Appendix E. We compare the memorability of these five sets of passphrases using CER and strength using guessrank.

Test sample. We generated one million passphrases from template-based diceware, and following the same distribution of lengths, we generated same number of passphrases from each of Diceware, Markov and MASCARA. We use these as test sample in our evaluations. For User, owing to the limited size of the dataset, we used only 6,500 user passphrases as our test sample. Note that the length distribution of User passphrases is different from others, as users often tend to utilize passphrases of smaller lengths. We did not use the length distribution of User for the system generated samples for this reason—as that would bias the test samples from Diceware, Markov and MASCARA towards smaller length passphrases.

5.1 Strength of the passphrases

We measure the strength of a passphrase based on their guessrank using the *min auto* approach discussed in Section 4.2. Note that we considered an offline generalized untargeted adversary as mentioned in Section 2.2. To compute a min-auto rank of password we take the minimum guessrank according to a number of guessrank estimations. Prior work has shown such approach provides close approximation of real-worlds.

We considered seven guessing algorithms for the min-auto approach: 2-gram and 3-gram word-based Markov models and 4-gram, 5-gram, 6-gram character-based Markov model [45], Wiki-5 bigram model, and template-based guessing algorithm. We explain last two guessing models below.

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

Recall that according to the threat model described in Section 2.2, the attacker has access to a huge corpus of passphrases generated from the system whose passphrases are being cracked. Therefore, the attacker have 10^7 system-generated passphrases from {Diceware, Markov, Mascara, TemplateDice }, on which the attacker can train his cracking algorithm. For User passphrases, the attacker trains on a smaller set of 70 thousand passphrases.

This trained model is then used to guess the passphrases of the corresponding test samples. Each passphrase has a probability of generation according to a model and using the method in Section 4.2, we can estimate the guessrank that is, the number of guesses the attacker will need to guess the passphrase correctly using that particular guessing model. A smoothing factor is used for any out of vocabulary (OOV) n-grams encountered. We took the minimum guessrank across all the models as the final output.

Results. We show the (estimated) guessranks of the passphrases in the test samples in Figure 2. Diceware passphrases are the most secure with 50% of passphrases requiring at least 10^{40} guesses. On the other end of the spectrum, we have User passphrases, with 50% of passphrases guessed within 10^{14} guesses, which is not even as secure as some of the most secure passwords [60]. The predictability

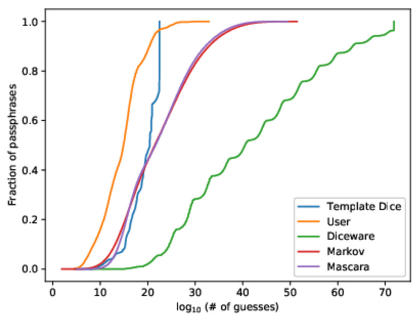

Figure 2: Comparing the strength of different sets of passphrases by evaluating their guessrank. Here, a cumulative distribution frequency of log guessrank (base 10) is shown. We can see that diceware passphrases are the most secure, while the user ones being the most predictable.

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 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 40th percentile. These are the passphrases of a length less than 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 improves upon it and requires 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 3. 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.

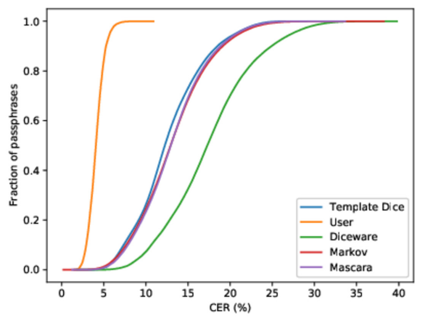

Figure 3: Distribution (CDF) of CER (Character Error Rate) for passphrases among the various samples. Diceware passphrases have the highest CER, making it the least memorable and User passphrases have the lowest CER which in turn means that they are easiest to remember.

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.

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

Although we would like to increase the guessrank (decrease the guessability) 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 show that MASCARA mitigate the limitations of TemplateDice discussed in Section 3.3 and is potentially much more effective to use in practice. 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 significant personalization.

6 USER STUDY

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

Ethical considerations. 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 [3]). The study was performed with informed consent. We asked 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, reducing privacy risk to the participants.

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 4: #participants who completed Part 1 and returned for Part 2 with return rate (across different algorithms).

6.1 Design and setup

Our two-part 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 uniformly choosing passphrase length from one of three length ranges (≤ 7, 8-12, >12). We used the range of ≤ 7 as most of user-generated passphrase (in our previous dataset from Section 3.3) 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 [65].

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. 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. Then, 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 distract participants before asking for authentication, the participants answered demographics, some generic questions and attention check questions.

Part 2. We invited participants who successfully finished part 1 to return and authenticate again after 48 hours from their completion time. A two-day recall time interval is used and justified for most prior password and passphrase recall research [29, 32, 36, 65]. In fact Huh et al. argued that a good recall rate after two days potentially (empirically) signifies the practically required memorability of passphrase—thus we used the same interval [29]. However, 24% of the users responded to the second part of the survey post 96 hours (4 days or more). 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 F. For analysis, we used Kruskal-Wallis (KW) test and pairwise Mann-Whitney U tests to find statistically significant differences at $\alpha = 0.05$ [41]. Our registration and login prompts are designed to reflect a real login system in line with a plethora of previous work [18, 59, 65]. Like earlier work on conducting ecologically valid password study [18], we did not nudge the user to choose/remember passphrases for any specific system (e.g., Diceware or Mascara). Thus, we strongly believe our data (e.g., recall) also captures users' ecologically valid preferences.

6.2 Demographics and participants info

A total of 310 Prolific users participated in our user study (we also ran 5 pilots in the survey development phase and updated questions to remove ambiguity according to pilot feedback). Among these participants, we detected invalid responses from 9 participants (2.9%) who failed the attention check. After excluding those, 301 participants successfully chose the passphrases, practiced them, and answered all the survey questions properly.

Forty-eight hours after Part 1, we emailed participants to return for the authentication in the second phase. Out of 301 participants, 217 participants returned within 24 hours of sending email, yielding a return rate of 72.1%. Out of these participants, 202 (93.1%) self-reported that their desired passphrase length (in number of words) is 7 or less for being able to remember in daily use.

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.

Return rate is high for Mascara due to better recall: The distribution of our participants across the different models and their return rate is also shown in Figure 4 (demographics are similar). The return rate is very high in Mascara, almost 25% higher than Diceware. To further investigate, we checked the Part 1 recall rates (posted right after practicing 5 times) between the users who returned in part 2 and those who did not. We omit the detailed results for brevity, but we found that across all algorithms, users who did not return in part 2 has a significantly low part 1 recall rate. This difference is more prominent for passphrases with length 7 or less. E.g., for that passphrase length Mascara have a recall rate of 92.31% in ones who returned for part 2 and 60% for those who did not. These results hint that the low return rate for Diceware and Markov is indicative of the underlying fact that a significant fraction cannot recall the passphrases. In fact one participant mentioned for a Dice passphrase that *"It was the longest and used random words"* and another mentioned for a Markov passphrase that *"I'm already pretty confident I'm not gonna remember this one guys :("*. These results indicate that some Diceware and Markov passphrases are hard for users to remember.

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

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, respectively. We perform a KW test to see how different the underlying distributions are. 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 gives $p = 0.39$ ($H = 2.97$), identifying that the underlying distribution is not significantly different among part 2 participants.

Passphrase strength. We estimate the strength of the passphrases by their guess rank, which was calculated apriori using the process mentioned in Section 5.1. The mean log guess rank (base 10) across Mascara, TemplateDice, Markov, and Diceware are 14.80, 11.70, 14.46, and 36.49, respectively. Similarly, the median log guess rank (base 10) across Mascara, TemplateDice, Markov, and Diceware are 14.20, 12.51, 12.86, and 36.05, respectively. Thus, for non statistically different length distributions, the Diceware and TemplateDice are most and least secure, respectively.

To compare the underlying distribution of the guess rank among the four algorithms, we again perform the KW test. This KW test rejects the null hypothesis ($H = 157.7, p \approx 0$), confirming that the underlying distributions are statistically significantly different. We then perform the Mann-Whitney U test on all possible pairs, and further find that guess ranks of *each pair* to be statistically significantly different. Next, we check the memorability of these passphrases using survey responses.

6.4 Evaluating passphrase memorability

Recall and CER. In our case, *successful recall* and *low CER* signify high memorability. Recall is successful if in part 2 users correctly input every character (including spaces) of the passphrase within five attempts. Also, the character error rate (CER) for a particular attempt is the edit distance between the attempt and the original passphrase divided by the number of characters in the original passphrase. We calculate CER for a user as the minimum CER across all attempts. In part 1 (right after seeing the passphrase) Mascara, TemplateDice, Markov, and Diceware have similar recall

Model	Recall	Mean CER	Median CER
Mascara	46.15%	19.10%	8.82%
TemplateDice	28.57%	30.03%	34.48%
Markov	14.29%	39.74%	43.59%
Diceware	23.08%	45.48%	56.41%

Figure 6: % participants who were shown passphrase length ≤ 7, with successful recall, mean and median CER (as %) while authenticating after 2 days. Recall of Mascara is 2x of that of Diceware and 1.6x of TemplateDice.

rates (between 50% to 60%) overall. However, for passphrases of length 7 or less (self-reported as preferred by more than 90% participants), Mascara has a Part 1 recall rate of 83.3%, significantly outperforming other algorithms (60%—65% for this length range).

Recall after two days. Figure 5 shows the percentage of users who were able to successfully recall passphrases chosen in part 1 (recall rate), as well as the mean and median CER for all users after 2 days. Mascara has the highest recall rate of 26.23% among all algorithms (TemplateDice recall only 17.46%). Surprisingly, the recall rate of Diceware is 24%, not too far from Mascara.

However, when we investigated further, we made two key observations. First, the return rate of Mascara was 84.72% and that of Diceware was only 60.97%. As we discussed before, the recall rate in part 1 for users who did not return in part 2 was significantly lower than the ones who returned. So, we already have close to 40% users from part 1 who did not remember Diceware passphrase (as opposed to less than 16% for Mascara).

Second, for participants who returned for part 2, Figure 6 shows the recall rate, mean and median CER for passphrases of length 7 or less, which the majority of participants wanted to use. We hypothesize that for remembering higher-length passphrases, there are other confounding factors including strong user bias against using very long phrases for day to day use. For passphrases of length 7 of less, Mascara have a recall rate of 46.15% after two days of no forced practice and less than 10% median CER. This recall rate is 2x higher than Diceware and 1.6x higher than TemplateDice, which are used in real-world systems. Furthermore, the median character error rate (CER) for Mascara passphrases (8.82%) is 6-times lower than Diceware (56.41%) when the passphrases of length seven or less are considered. We further checked if the edit distances between typed passphrases and actual passphrases for Mascara is statistically significantly lesser than other algorithms (lesser edit distance impliers lesser error and more memorability). We ran pairwise Mann-whitney U tests (with Bonferroni correction for multiple tests) over the edit distances for these passphrases in part 2. The edit distances for Mascara passphrases is indeed statistically significantly lower than other algorithms ($p < 0.005$). In fact the average edit distance for Mascara is 6.27 at the end of part 2 which is approximately 2.9 times less than Diceware (18.61). So, for the desired length of passphrases (as self-reported by the users) Mascara significantly improves the memorability of state of the art passphrase generation algorithms.

MASCARA is not a human-in-the-loop (HITL) approach. However, some prior work took a HITL approach of passphrase generation too. Just to test the utility of MASCARA (even though it's not a fair comparison) we compare MASCARA with a guided word choice (GWC) method by Blanchard et al. [5] using a separate user study (Appendix G). Although GWC provides users more control for choosing their random words, the user study reveals that Mascara-passphrases are still at least comparable or a little more memorable (and less error prone) than GWC-generated passphrases.

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 [9]. Second, we show 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 and culture-sensitive passphrases which is a fertile avenue for future work.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Hyoungshick Kim for their valuable feedback. We also thank Abir De, Mohit Gupta and Husne Atiya for their help and input with an earlier iteration of this work. This research was (partially) funded by the Federal Ministry of Education and Research (BMBF), Germany under the project LeibnizKILabor with grant No. 01DD20003 and by a Google India Faculty Research Award.

REFERENCES

[1] AgileBits. 2018. 1Password passphrase generation. `https://github.com/1Password/spg`.

[2] Mahdi Nasrullah Al-Ameen, Matthew Wright, and Shannon Scielzo. 2015. Towards making random passwords memorable: Leveraging users' cognitive ability through multiple cues. In *CHI'15*. 2315–2324.

[3] Tom L Beauchamp. 2008. The belmont report. *The Oxford textbook of clinical research ethics* (2008), 149–155.

[4] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: Analyzing text with the natural language toolkit.*

[5] Nikola K. Blanchard, Clément Malaingre, and Ted Selker. 2018. Improving Security and Usability of Passphrases with Guided Word Choice. In *ACSAC '18*. 723–732.

[6] Joseph Bonneau. 2012. Statistical Metrics for Individual Password Strength. In *Security Protocols Workshop.*

[7] Joseph Bonneau. 2018. EFF's new wordlists for random passphrases. `https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases`

[8] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. 2012. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE S&P*. 553–567.

[9] Joseph Bonneau and Ekaterina Shutova. 2012. Linguistic Properties of Multi-word Passphrases. In *FC*, Jim Blyth, Sven Dietrich, and L. Jean Camp (Eds.). 1–12.

[10] Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. 1992. An Estimate of an Upper Bound for the Entropy of English. *Comput. Linguist.* 18, 1 (1992), 31–40.

[11] Julio Casal. 2019. 1.4 Billion Clear Text Credentials Discovered in a Single Database. `https://bit.ly/3r512M7`.

[12] Rahul Chatterjee, Anish Athayle, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. 2016. pASSWORD tYPOS and how to correct them securely. In *IEEE S&P*. 799–818.
[13] Rahul Chatterjee, Joanne Woodage, Yuval Pnueli, Anusha Chowdhury, and Thomas Ristenpart. 2017. The typtop system: Personalized typo-tolerant password checking. In *CCS'17*. 329–346.
[14] Catalin Cimpanu. 2020. FBI recommends passphrases over password complexity. https://zd.net/38Qlwzr.
[15] Cristian Danescu-Niculescu-Mizil, Justin Cheng, Jon Kleinberg, and Lillian Lee. 2012. You had me at hello: How phrasing affects memorability. In *ACL'12*.
[16] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The tangled web of password reuse.. In *NDSS'14*, Vol. 14. 23–26.
[17] Matteo Dell'Amico and Maurizio Filippone. 2015. Monte Carlo strength evaluation: Fast and reliable password checking. In *CCS'15*. 158–169.
[18] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. 2013. Does My Password Go up to Eleven? The Impact of Password Meters on Password Selection. In *CHI'13*. 2379–2388.
[19] Enpass. 2023. Enpass passphrase generation. https://www.enpass.io/password-generator/.
[20] Shayan Eskandari, Jeremy Clark, David Barrera, and Elizabeth Stobert. 2018. A first look at the usability of bitcoin key management. *arXiv preprint arXiv:1802.04351* (2018).
[21] Xianyi Gao, Yulong Yang, Can Liu, Christos Mitropoulos, Janne Lindqvist, and Antti Oulasvirta. 2018. Forgetting of passwords: ecological theory and data. In *USENIX SEC'18*. 221–238.
[22] Wolf Garbe. 2022. Symspell. https://github.com/wolfgarbe/SymSpell.
[23] GeoNames. 2005. Major cities of the world. http://www.geonames.org.
[24] Marjan Ghazvininejad and Kevin Knight. 2015. How to memorize a random 60-bit string. In *NAACL'15*. 1569–1575.
[25] Murray Grant. 2014. Template based diceware algorithm. https://github.com/ligos/MakeMeAPassword.
[26] Paul A Grassi, Michael E Garcia, and James L Fenton. 2017. DRAFT NIST special publication 800-63-3 digital identity guidelines. *NIST* (2017).
[27] Yimin Guo, Zhenfeng Zhang, and Yajun Guo. 2019. Optiwords: A new password policy for creating memorable and strong passwords. *Computers & Security* 85 (2019), 423–435.
[28] hashcat Wiki. 2023. Example hashes - Generic hash types. https://hashcat.net/wiki/doku.php?id=example_hashes
[29] Jun Ho Huh, Seongyeol Oh, Hyoungshick Kim, Konstantin Beznosov, Apurva Mohan, and S. Raj Rajagopalan. 2015. Surpass: System-Initiated User-Replaceable Passwords. In *CCS'15*. 170–181.
[30] Alexa Huth, Michael Orlando, and Linda Pesante. 2012. Password security, protection, and management. *US Computer Emergency Readiness Team* (2012).
[31] Noopa Jagadeesh and Miguel Vargas Martin. 2021. Alice in Passphraseland: Assessing the Memorability of Familiar Vocabularies for System-Assigned Passphrases. arXiv:2112.03359 [cs.CR]
[32] Zeinab Joudaki, Julie Thorpe, and Miguel Vargas Martin. 2018. Reinforcing system-assigned passphrases through implicit learning. In *CCS'18*. 1533–1548.
[33] Marcel Adam Just and Patricia A. Carpenter. 1980. A theory of reading: from eye fixations to comprehension. *Psychological review* 87 4 (1980), 329–54.
[34] Keepass. 2023. Keepass passphrase generation. https://keepass.info/plugins.html#ppgen.
[35] Mark Keith, Benjamin Shao, and Paul Steinbart. 2009. A Behavioral Analysis of Passphrase Design and Effectiveness. *JAIS* 10 (02 2009), 63–89.
[36] Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy M. Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio López Hernandez. 2012. Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms. *IEEE S&P*, 523–537.
[37] Daniel V Klein. 1990. Foiling the cracker: A survey of, and improvements to, password security. In *2nd USENIX Security Workshop*. 5–14.
[38] Per Ola Kristensson and Keith Vertanen. 2012. Performance comparisons of phrase sets and presentation styles for text entry evaluations. In *IUI'12*. 29–32.
[39] Cynthia Kuo, Sasha Romanosky, and Lorrie Faith Cranor. 2006. Human selection of mnemonic phrase-based passwords. In *SOUPS'06*. 67–78.
[40] Kok-Wah Lee and Hong-Tat Ewe. 2007. Passphrase with semantic noises and a proof on its higher information rate. In *CISW 2007*. 652–655.
[41] E. L. Lehmann and Joseph P. Romano. 2005. *Testing statistical hypotheses* (third ed.). Springer. xiv+784 pages.
[42] Luis A Leiva and Germán Sanchis-Trilles. 2014. Representatively memorable: sampling the right phrase set to get the text entry experiment right. In *CHI'14*. 1709–1712.
[43] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. 2019. Protocols for checking compromised credentials. In *CCS'19*. 1387–1403.
[44] Yi Liu, Ruilin Li, Xingtong Liu, Jian Wang, Lei Zhang, Chaojing Tang, and Hongyan Kang. 2017. An efficient method to enhance Bitcoin wallet security. In *ASID'17*. 26–29.
[45] Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. 2014. A Study of Probabilistic Password Models. In *IEEE S&P*. 689–704.
[46] I Scott MacKenzie and R William Soukoreff. 2002. A character-level error analysis technique for evaluating text entry methods. In *NordCHI'02*. 243–246.
[47] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2016. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In *USENIX SEC'16*. 175–191.
[48] OpenAI. 2019. Better Language Models and Their Implications. https://openai.com/blog/better-language-models/.
[49] OpenSSH. 2013. ssh-keygen(1) - Linux man page. https://linux.die.net/man/1/ssh-keygen.
[50] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. 2019. Beyond credential stuffing: Password similarity models using neural networks. In *IEEE S&P*. 417–434.
[51] pcmag. 2022. Survey of password managers. https://bit.ly/3ukEWKf. Accessed: 2022-01-31.
[52] P.Norvig. 2012. English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU. http://norvig.com/mayzner.html.
[53] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[54] AG Reinhold. 1995. The Diceware Passphrase Home Page. https://theworld.com/~reinhold/diceware.html.
[55] Alfréd Rényi et al. 1961. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*.
[56] Sena Sahin and Frank Li. 2021. Don't Forget the Stuffing! Revisiting the Security Impact of Typo-Tolerant Password Authentication. In *CCS'21*. 252–270.
[57] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2012. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *SOUPS'12*. 1–20.
[58] Elizabeth Stobert and Robert Biddle. 2014. The password life cycle: user behaviour in managing passwords. In *SOUPS'14*. 243–255.
[59] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2012. How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation. In *USENIX SEC'12*. 65–80.
[60] Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. 2015. Measuring Real-World Accuracies and Biases in Modeling Password Guessability. In *USENIX SEC'15*. 463–481.
[61] Steven Van Acker, Daniel Hausknecht, Wouter Joosen, and Andrei Sabelfeld. 2015. Password Meters and Generators on the Web: From Large-Scale Empirical Study to Getting It Right. In *CODASPY'15*.
[62] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. 2016. Targeted online password guessing: An underestimated threat. In *CCS'16*. 1242–1254.
[63] Grady Ward. 2002. Moby Word Lists by Grady Ward. http://www.gutenberg.org/ebooks/3201.
[64] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. 2010. Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords. In *CCS'10*. 162–175.
[65] Simon S Woo. 2020. How Do We Create a Fantabulous Password?. In *WWW'20*. 1491–1501.
[66] Simon S Woo and Jelena Mirkovic. 2016. Improving recall and security of passphrases through use of mnemonics. In *10th International Conference on Passwords (Passwords'16)*.
[67] J. Yan, A. Blackwell, R. Anderson, and A. Grant. 2004. Password memorability and security: empirical results. *IEEE Security Privacy* 2, 5 (2004), 25–31.

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 [11], 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 released the code/dataset from this paper for further research on passphrases in

https://github.com/Mainack/MASCARA-passphrase-code-data

Password leak dataset. We use a compilation of prior data breaches that surfaced in 2018 by 4iQ security firm [11]. The leaked dataset contains nearly 1.4 billion email-password pairs. Prior research on passwords has used this leak [43, 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 [52]). 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 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 passwords and creating passphrases. Initially we tried segmenting using SymSpell library [22] which is parameterized by a unigram distribution V (created using popular word lists [4, 23, 63]). 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. However, this segmentation often failed to identify proper nouns like city names or first names which are often part of a password. Thus, we devised a hybrid approach. First, we implemented a greedy strategy to find known words in a password where we searched for words from exhaustive lists of known words including names of countries and popular cities [23], and common first names in the US [63]. We only consider a password as passphrase if it has at least three valid English words, each of length greater than or equal to 3. If this greedy approach fails to segment a given passphrase (likely signifying variations of words), we used SymSpell library for segmentation.

Resulting passphrases dataset. Using our segmentation approach, we found 72, 999 passphrases from our segmentation algorithm. 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 7.

[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`
	`get-there-very-fast-indeed`

Unpopular passphrases	`friendly-neighborhood-pickle`
	`eddie-the-penguin-stick`
	`super-looper-evil-ben`

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

Common names	`KatherineCarrasquillo`
	`zuleimahernandez1230`
	`dobrovolskayatatiana`

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

Figure 7: 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 checked the passwords which were not considered as passphrases by our algorithm to gauge the false negative rate. We random sampled hundred such passwords from 1.8 million passwords that are discarded by our algorithm and manully analyzed them to find three key types in this set: (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 at the bottom three groups of rows in Figure 7.

A.2 Ecological validity of our dataset

As mentioned earlier, the passphrases we identified are a subset of a broader password breach dataset that is widely used in prior works and is known to contain real email-password pairs [11]. According to the work by Bonneau et. al. [9], passphrases in their Amazon PayPhrase dataset (not publicly available) have linguistic properties similar to natural language. We make the same observation in our dataset too using a GPT LM-scorer, which shows user-generated passphrases that we extract are similar to natural text (Figure 8) and some examples are given in Figure 7. We verified that the frequency distribution of the passphrase dataset mirrors that of the original breached password dataset, hinting at ecological validity of our passphrase dataset–three most frequent passphrases are used by 252 (0.26%), 115 (0.12%), and 102 (0.10%) users.

B LINGUISTIC PROPERTIES OF IN-USE PASSPHRASES

We leveraged a natural language model GPT-2 [53] to check similarity of various system-generated passphrases as well as user passphrases to the natural language [53]. GPT-2 is trained to predict the next word given a sequence of words and widely used

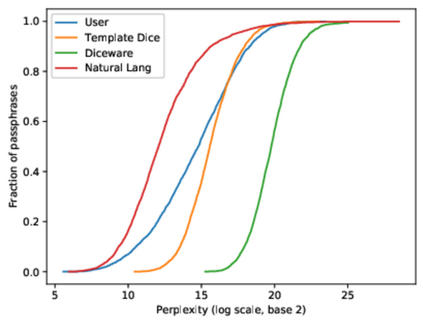

Figure 8: CDF of logarithm of perplexity to the base 2 for Diceware, TemplateDice and User passphrases, along with natural language phrases for baseline.

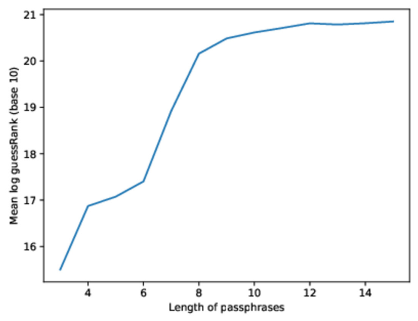

Figure 9: Variation of mean $\log_{10}$ guessrank across length for the TemplateDice passphrases. We observe that guessranks reached a plateau after length eight.

today [48]. We sampled 3000 passphrases of similar length distribution from each system (dataset for User) and compute their perplexity using GPT-2. Lower perplexity implies similarity to natural language. The distribution of the perplexity score [10] for both the system (Diceware and TemplateDice) and User passphrases are shown in Figure 8. User passphrases have a very low perplexity value which is similar to the perplexity of a natural language corpus, uncovering 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.

C 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 of TemplateDice

D TRADE-OFF FOR MASCARA PARAMETERS

Constraint thresholds. We introduce θ_1 and θ_2 as thresholds to be satisfied by each of the probable words from the support (at every step of generation). Words that fail in any of the two constraints are removed from the support, and then we choose the next word weighted by their conditional bigram probabilities.

The goal of these empirical thresholds is 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, $L_2(w_{i-1}, w_i)$. They also ensure that generated passphrases can still retain syntactic structure and flow.

Since our CER estimate is a linear fit, and the log probability of a phrase is sum of the log probabilities of its constituent unigrams or bigrams, we estimated 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 for using a different corpus or controlling 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 intermediate CER score (S), θ_1, is set as fraction of maximum CER.

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`
GWC	1)	`rub revisions lilo clerk apple beting`
	2)	`helios hounds binary canonized lady overflight`
	3)	`pressures broth billable playgirl raita dekko`

Figure 10: Three randomly sampled passphrases from each group of passphrases we consider for evaluation.

E EXAMPLE OF PASSPHRASES FROM DIFFERENT ALGORITHMS

Figure 10 presents a set of randomly chose passphrases generated by each algorithm and the ones generated by users.

F USER STUDY INSTRUMENT

F.1 Part 1

Informed consent

First we show the informed consent to the users. Users will see the questions below only if they indicated that they understood the requirements and agree to participate.

- Please enter your prolific id ______
- When you login to your online accounts you often need a credential, i.e., your email/username and a secret.

Two possible ways is creating the secret is: choosing a password (a set of characters) or choosing a passphrase (a set of words). Example of a password is "!Passw0rd!" and example of a passphrase is "correct horse battery staple". You can also use a password or a passphrase as a master secret for accessing your password manager (a software that stores and manages all of your login credentials across multiple online accounts).
Do you use passphrases as a secret for logging-in to any of your online accounts? ○ Yes ○ No
if YES to use passphrases

- Please briefly explain why do you use passphrases for these account(s) instead of passwords? (1-2 sentences) ______
  if YES to use passphrases
- 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
  ○ Other: ______
  if NO to use passphrases
- Briefly explain why do you NOT use passphrases for these account(s) (and use passwords)? (1-2 sentences)
  if NO to use passphrases
- If you have to use a passphrase as your master credential for a password manager, how would you generate it? Choose all that apply.
  ○ Use an online tool [also write names, if you remember]: ______
  ○ Self-generated - will use a quote from a poem, movie, or book
  ○ Self-generated - will use a combination of random words
  ○ Other: ______
- Which of these crawls? □ Dog □ Cat □ Snake □ Kite
- For 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

  Passphrase choice
- In this section, we will show you a few passphrases and ask you to choose one. A good passphrase should be long so that others cannot guess it, but also be easy to remember so that you can enter the passphrase with minimum errors.
  We are showing three passphrases and how secure they are in terms of the time it might take to guess the passphrase by an attacker. Please choose one passphrase which you prefer as your master login credential (e.g., for your password manager).
  Please do not write down the chosen passphrase.Try to remember it to the best extent possible.
  Please DO NOT COPY/PASTE passphrases in this study. Such actions will be detected and your task could be invalidated.
  [Passphrase 1] [Passphrase 2] [Passphrase 3]
  Repeat question below 5 times for practicing
- For practicing, please enter your chosen passphrase: [CHOSEN PASSPHRASE] (4 more practices remaining)
  Post Passphrase choice questions
- Why did you choose this particular passphrase among the ones shown? (1 - 2 sentences): ______
- 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: ______
  Demographics
- Which age group do you belong to? ○ 18-30 ○ 31-45 ○ 46-60 ○ 60+
- Which gender do you identify yourself most with? ○ Male ○ Female ○ Non-Binary / Third Gender ○ Prefer not to say
- What is the highest degree or level of school you have completed? ○ Some high school ○ High school ○ Some college ○ Trade, technical, or vocational training ○ Associate's degree ○ Bachelor's degree ○ Master's degree ○ Professional degree ○ Doctorate ○ Prefer not to say
- 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.

F.2 Part 2

Welcome to a brief follow-up of the earlier study on passphrases that you completed. Recall that our study is on understanding utility of passphrases as login credentials for online accounts.

Your primary task in this final part of this study is to just re-enter your chosen passphrase as you remember in this survey and answer a few questions regarding your current perception about passphrases. This part will take around 3 to 5 minutes of your time. You'll be compensated $1.00 USD for this part. Thank you for helping in our research to improve the privacy and security of users by understanding usage of passphrases. **Questions**

- To start the survey please enter your Prolific ID
  Check recall after 2 days
  Repeat the question below at most five times
- Please enter the passphrase you chose in Part 1 of this study below. (5 tries remaining). ______
- 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
- Please briefly explain your choice of preferred length for passphrases (1-2 sentences): ______
- 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
- 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 don't want to change the passphrase.
  ○ Yes, I want to change my chosen passphrase to ______
- In brief, why do you feel your modification to the chosen passphrase will make the passphrase more memorable without making it easy to guess for others? (1-3 sentences) ______
- 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
- Please briefly explain your response (1-3 sentences) ______

G COMPARING MASCARA WITH HUMAN-IN-THE-LOOP PASSPHRASE GENERATION

We further tested the utility of Mascara against passphrase generation systems which puts human in the loop (and presumably incur higher cognitive and time cost for users in exchange for more user control). Specifically, we compared the memorability of Mascara generated passphrases with passphrases generated via guided word choice (GWC) method [5]. GWC asks users to choose words for their passphrase from a list of random words shown to them. On our request, Blanchard et al. graciously shared the set of passphrases used in their experiment [5] (Implementation of GWC is unavailable). We show a sample of GWC passphrases in Figure 10. Thus, in this experiment we repeated part 1 of our original study (since Blanchard et al. also did not ask participants for a followup task [5]). A total of 37 users (recruited via mailing lists) were randomly shown either Mascara generated passphrase or GWC-generated passphrases (from original paper). Then these users typed the chosen passphrases via a login screen after answering a few distracting questions. (18 users were shown GWC passphrases and 19 were shown Mascara passphrases). A comparable 72.2% users recalled GWC passphrases correctly in one try and 73.7% users recalled Mascara passphrases in one try. However, interestingly, the average edit distance for GWC passphrases was 2.72 which is higher than the average edit distance of Mascara passphrases (1.46). Thus, although GWC provides users more control while choosing random words, in our experiments Mascara-passphrases are still slightly more memorable (and less error prone) than GWC-generated passphrases.