Improving Password Guessing via Representation Learning

Dario Pasquini[1,2,3], Ankit Gangwal[1,4], Giuseppe Ateniese[1], Massimo Bernaschi[3], and Mauro Conti[4]

[1]Stevens Institute of Technology, USA
[2]Sapienza University of Rome, Italy
[3]Institute of Applied Computing, CNR, Italy
[4]University of Padua, Italy
{dpasquin, gatenies, agangwal}@stevens.edu, massimo.bernaschi@cnr.it, conti@math.unipd.it

Abstract

Learning useful representations from unstructured data is one of the core challenges, as well as a driving force, of modern data-driven approaches. Deep learning has demonstrated the broad advantages of learning and harnessing such representations.

In this paper, we introduce a GAN-based representation learning approach for password guessing. We show that an abstract password representation naturally offers compelling and versatile properties that can be used to open new directions in the extensively studied, and yet presently active, password guessing field. These properties can establish novel password generation techniques that are neither feasible nor practical with the existing probabilistic and non-probabilistic approaches. Based on these properties, we introduce: (1) A framework for password guessing for practical scenarios where partial knowledge about target passwords is available and (2) an Expectation Maximization-inspired framework that can dynamically adapt the estimated password distribution to match the distribution of the attacked password set, leading to an optimal guessing strategy.

1 Introduction

Text-based passwords remain the most common form of authentication, as they are both easy to implement and familiar (to users) . However, text-based passwords are vulnerable to guessing attacks. These attacks have been extensively studied, and it is still an active area of research. Modern password guessing attacks are founded on the observation that human-chosen passwords are not uniformly distributed in the password space (i.e., all possible strings). This is due to the natural preference of choosing (easily-)memorable passwords that cover only a small fraction of the exponentially large password space. As a consequence, real-world password distributions are typically composed by few dense zones that can be feasibly estimated by an adversary to perform password-space reduction attacks [59]. Along that line, several probabilistic approaches have been proposed [28, 42, 57]. These techniques - under different assumptions - try to directly estimate the probability distribution behind a set of observed passwords. Such estimation is then used to generate suitable guesses and perform efficient password guessing attacks.

Figure 1: A small section of the induced latent space around the latent point for the password “rockyou”.

Orthogonal to the current lines of research, we demonstrate that an adversary can further expand the attack possibilities by leveraging representation learning techniques [19]. Representation learning aims at learning useful and explanatory representations [19] from a massive collection of unstructured data. By applying this general approach on a corpus of leaked passwords [15], we demonstrate the advantages that an adversary can gain by learning a suitable representation of the observed password distribution; rather than directly estimating it. In this paper, we show that such password representation indeed permits an attacker to establish novel password guessing techniques that further jeopardize password-based authentication

systems.

Inspired by the recently introduced deep learning-based approach for password guessing, i.e., PassGAN [35], we choose to model the representation of passwords in the latent space of an instance of Generative Adversarial Networks (GAN) [30] generator. This representation, thanks to its inherent smoothness [19], is able to enforce a semantic organization in the high-dimensional password space. Such an organization mainly implies that respective representations of semantically-related passwords are closer in the latent space of the generator. As a consequence, geometric relations in the latent space directly translate to semantic relations in the data space. A representative example of this phenomenon is loosely depicted in Figure 1, where we show some latent points (with their respective plain-text passwords) localized in a small section of the induced latent space.

We can exploit such geometric relations to perform a peculiar form of conditional password generation. In the study of such relations, we characterize two main properties, namely, *password strong locality* and *password weak locality*. These locality principles enforce different forms of passwords organization that allow us to design two novel password guessing frameworks, *Substring Password Guessing (SSPG)* and *Dynamic Password Guessing (DPG)*. We highlight that the state-of-the-art approaches are either unable or inefficient to perform such type of advanced attacks. The major contributions of our work are as follows:

1. We are the first to demonstrate the potential of using fully unsupervised representation learning in the domain of password guessing.

2. We introduce a probabilistic and completely unsupervised form of template-based passwords generation. Using this technique, we build a practical framework that is able to efficiently perform targeted password guessing in the presence of partial knowledge. We call this framework SSPG. SSPG can be used: (1) by an adversary to increase the impact of side channels and similar password attacks [16, 18, 41, 56]; or (2) by a legitimate user to recover his/her password. We show the efficiency of SSPG with respect to its direct competitors via experimental evaluations.

3. We introduce the concept of DPG: DPG is the password guessing approach that dynamically adapts the guessing strategy based on the feedback received from the interaction with the attacked passwords set. We build an Expectation-Maximization inspired DPG implementation based on the principle of password weak locality. DPG shows that an attacker can consistently increase the impact of the attack by leveraging the passwords guessed during a running attack.

It is important to highlight that these properties, and their distinctive capabilities, come practically for **free** with the latent representation learned by the underlying deep generative model. In addition, the ongoing continuous developments in the GAN framework would naturally further improve our approaches.

Organization: Section 2 gives an overview of the fundamental concepts related to our work. Here, we also present our model improvements and the tools upon which our core work is based. We present password strong locality along with SSPG in Section 3 and password weak locality along with DPG in Section 4. The evaluation of our proposed techniques is presented in their respective sections. Section 5 briefly discusses relevant previous works. Section 6 concludes the paper, although supplementary information are provided in Appendices.

2 Background and preliminaries

In Section 2.1, we explain GAN and related concepts that are important to understand our work. Section 2.2 briefly discusses the technical aspects of PassGAN, which is the closest work to ours. In Section 2.3, we present our model improvements and the tool that is a fundamental building block in our approach.

2.1 Generative Adversarial Network (GAN)

GAN is a framework to train a parametric probabilistic model to perform implicit estimation of an unknown **target data distribution** $p^*(x)$, for a given observed random variable $\mathbf{x}$ [29, 30].

In contrast to the common *prescribed probabilistic models* [25], implicit probabilistic models do not explicitly estimate the probability distribution; they instead approximate the stochastic procedure that actually generates data [43]. In other words, we can sample data points from the model as if they were sampled from a random variable following $p(\mathbf{x})$. Although, we cannot directly compute the probability of a given state x_i of $\mathbf{x}$.

This class of models is capable of successfully representing data distribution defined in a very high dimensional space, such as in the case of images [21]. GAN generators have established the new state-of-art in several generative tasks [21, 36, 60].

The parametric function used for the estimation is a deep neural network defined following an adversarial training approach. The latter process is guided by a second network D (i.e., the critic/discriminator), which gives a density estimation-by-comparison [43] loss function to the generative model G (i.e., the generator). The adversarial training bypasses the necessity of defining an explicit likelihood-function and allows us to have a good estimation of very sharp distributions [29].

GAN generators are latent variables models. They assume that each observable data instance can be modeled by a set of k

latent variables. The learned generator acts as a deterministic mapping function $G : \mathbf{Z} \rightarrow \mathbf{X}$ between the latent space and the data space (i.e., where the observed data is defined). The assumed latent space is continuous $\mathbf{Z} : \mathbb{R}^k$, and its points (that we refer to as **latent points**) are distributed following a simple uninformative prior distribution $\dot{p}(\mathbf{z})$ that we refer to as **prior latent distribution**[1], where the semantic aspects of the latent variables are completely entrusted to the generator that learns them in an unsupervised way.

The probability distribution represented by the generator has the following form:

$$p(\mathbf{x}) = p(\mathbf{x};\theta)\dot{p}(\mathbf{z}), \tag{1}$$

where θ is the set of learnable parameters of the generator (i.e., primarily the weights of the neural network). Both k and $\dot{p}(\mathbf{z})$ can be arbitrary chosen and fixed before the training. They can be intended as hyper-parameters of the model. Typical choices for $\dot{p}(\mathbf{z})$ are $\mathcal{N}(0,\mathbf{I})$ or $U[0,1]$ [29].

Sampling points from the latent space according to $\dot{p}(\mathbf{z})$ and then mapping them in the data-space through the generator, is equivalent to sampling data points from the data space $\mathbf{X}$ according to $p(\mathbf{x})$, where p($\mathbf{x}$) is the approximation of the target probability distribution $p^*(\mathbf{x})$ estimated by the generator. During this operation, we can generally also consider an arbitrary $p(\mathbf{z})$ that can be different[2] from $\dot{p}(\mathbf{z})$. In the rest of this paper, we will refer to the probability density function $p(\mathbf{z})$ of the latent space with the general term of **latent distribution.**

To accomplish the generative task, the latent representation is modeled to be able to capture the posterior distribution of the underlying explanatory factors of the observed data [47]. Similar to the feature embedding techniques [31, 37], the latent representations of semantically bounded data points show strong geometric relations in the latent space [47]. As a result of these properties, such unsupervised learned representation is often used for several other external tasks [48–50].

2.2 PassGAN and PassGAN+

Hitaj et al. in their work PassGAN [35] demonstrated the application of deep generative models as an implicit estimator of password distribution. The capabilities of such models are a result of: (1) their adversarial training process; and (2) the high capacity deep neural networks used for function approximation. These characteristics enable the model to capture the long-tailed distribution of a real-world passwords leak and outperform in expressivity other state-of-the-art tools [35]. PassGAN harnesses a Wasserstein GAN with gradient penalty [33] and a residual-block-based architecture [34]. It assumes a latent space that has standard normal distribution as its prior latent distribution and dimensionality equal to 128. The model is trained using an 80-20% split of the well known RockYou [15] password leak and only passwords with 10 or fewer characters are considered. They obtained their final test-set by removing the duplicate passwords and the common passwords occurring in both train-set and test-set.

PassGAN, due to its inherent training instability, does not exploit the full potential of the deep generative models in the context of password guessing. We propose a series of improvements in Section 2.3 to overcome these limitations. We will use the improved model (PassGAN+) as the basis for our encoder.

2.3 Our improvements & tools

In Section 2.3.1, we propose our model improvements that allows us to outperform PassGAN in the task of password guessing. In Section 2.3.2, we present our encoder network that we use as a tool to learn the inverse mapping. Our core contributions are founded upon these improvements and tools.

2.3.1 Model improvements

The password guessing approach presented in PassGAN suffers from an inherent training instability [35]. Hence, the generator and the discriminator may not perform a sufficient number of training iterations. This may lead to a limited approximation of the target data distribution and reduced accuracy in the password guessing task. Training instability is a common hurdle for GAN frameworks [17]. The discrete representation of the strings (i.e., passwords) in the train-set[3] introduces strong instability for two main reasons: (1) The discrete data format is very hard to reproduce for the generator because of the final *softmax* activation function, which can easily cause numeric instability and a low quality gradient; and (2) The inability of the generator to fully mimic the discrete nature of the train-set makes it very easy for the critic to distinguish[4] between real and generated data. Hence, the critic can assign the correct *"class"* easily, leaving no room for the enhancement of the generator; especially in the final stages of the training.

To tackle the problems above, we apply a form of stochastic smoothing over the representation of the strings contained in the train-set. Moreover, Sønderby et al. [52] showed that adding noise to the input of the critic causes benefits to the training process. Hence, in contrast to the work in [48], we smooth the input of the critic instead of the output prediction. The smoothing operation consists of applying an additive

[1] The adjective "prior" refers to the fact that we assume the latent variables are initially distributed as $\dot{p}(\mathbf{z})$.

[2] At a cost of representing a distribution different from $p^*(\mathbf{x})$.

[3] Each string is represented as a binary matrix obtained by the concatenation of the one-hot encoded characters.

[4] We refer to the original GAN formulation, where the critic is intended to discriminate between the true and the fake data.

noise of small magnitude over the one-hot encoding representation of each character. The smoothing operation is governed by a hyper-parameter γ, which defines the upper-bound of the noise's magnitude. We empirically chose $\gamma = 0.01$ and re-normalize each distribution of characters after the application of the noise. This smoothing operation has a significant impact on the dynamics of the training allowing us to perform **30 times more** training iterations without training collapse [21]. We keep the general GAN framework mostly unchanged because of the excellent performance of the gradient penalty in *WGAN* [33].

With our improvements in the training process, we can exploit a deeper architecture for both the generator and the critic. We substitute the plain residual blocks with deeper residual bottleneck blocks [34] leaving their number intact. We find the use of batch normalization in the generator to be essential to increase the number of layers of the networks successfully. For precise information about the architecture, please refer to the work [34]. Additionally, we reduced the dimensionality of the latent space from 128 to 64 inducing an additional small increment in performance for the password guessing task[5].

The new architecture and training process are collectively referred to as PassGAN+. With it, we are able to learn a better approximation of the target password distribution, and consequently, obtain a significant improvement on the number of guessed passwords in the password guessing scenario. This observation is supported by the results reported in Table 1, where PassGAN and our model PassGAN+ (both trained on 80-20% train-test split) are compared over the test-set of RockYou dataset.

Table 1: The matched passwords by PassGAN and PassGAN+ over the RockYou test-set

Sample Size	PassGAN (%)	PassGAN+ (%)
$1 \cdot 10^7$	2.03	2.43
$1 \cdot 10^8$	6.72	8.81
$1 \cdot 10^9$	15.09	20.13
$1 \cdot 10^{10}$	26.03	34.76
$2 \cdot 10^{10}$	29.54	39.39
$3 \cdot 10^{10}$	31.60	42.09
$4 \cdot 10^{10}$	33.05	43.98
$5 \cdot 10^{10}$	34.19	45.43

In this paper, we use all of the improved settings described in this section for our GAN model.

2.3.2 Learning the inverse mapping

To fully exploit the properties offered by the learned latent representation of passwords, we need a way to efficiently explore the latent space. Therefore, our first interest is to understand the relation between the observed data (i.e., passwords) and their respective latent representations; in particular, their position within the latent space. A direct way to model this relation is to learn the inverse of the generator function $G^{-1} : \mathbf{X} \rightarrow \mathbf{Z}$. Actually, GANs, by default, do not need to learn those functions because that requirement is bypassed by the adversarial training approach. To do so, framework variations [26, 27] or additionally training phases [38] are required.

To avoid any source of instability in the original training procedure, we opt to learn the inverse mapping only after the training of the generator is complete. This is accomplished by training a third **encoder** network E that has an identical architecture as the *critic*, except for the size of the output layer. The network is trained to simultaneously map both the real (i.e., data coming from the train-set) and generated (i.e., data coming from G) data to the latent space. Specifically, the loss function of E is mainly defined as the sum of the two cyclic reconstruction errors over the data space. This is shown in the following:

$$\begin{aligned} L_0 &= \mathbb{E}_z[d(G(z), G(E(G_t(z))))], \\ L_1 &= \mathbb{E}_x[d(x, G(E(x)))]. \end{aligned} \tag{2}$$

In Eq. (2), the function d is the cross-entropy whereas x and z are sampled from the train-set and the prior latent distribution, respectively. The variable t in L_0 refers to the temperature of the final *softmax* layer of the generator. In Eq. (2), we do not specify temperature on a generator notation when it is assumed that it does not change during the training. The combination of these two reconstruction errors aims at forcing the encoder to learn a general function capable of inverting both the true and generated data correctly. As discussed in Section 2.3.1, the discrepancy between the representation of the true and generated data (i.e., discrete and continuous data) is potentially harmful for the training process. To deal with this issue, we anneal the temperature t in loss term L_0 during the training. We do that to collapse slowly the continuous representations of the generated data (i.e., the output of the generator) towards the same discrete representation of the real data (i.e., coming from the dataset). Next, an additional loss term, shown in Eq. 3, is added forcing the encoder to map the data space in a dense zone of the latent space (dense with respect to the prior latent distribution).

$$L_2 = \mathbb{E}_z[d(z, E(G(z)))]. \tag{3}$$

Our final loss function for E is reported in Eq. 4. During the encoder training, we use the same train-set that we used to train the generator, but, we consider only the unique passwords in this case.

$$L_E = \alpha L_0 + \beta L_1 + \gamma L_2. \tag{4}$$

The information about the hyper-parameters we used is listed in Table A.1 in Appendix A.

[5] We speculate that such performance increment is induced by the more compact and so smarter latent representation.

3 Passwords strong locality and SubString Password Guessing (SSPG)

In this Section, we present the first major contribution of our paper, i.e., the password strong locality concept and its possible applications for password guessing. In Section 3.1, we introduce the concept of password strong locality with the help of different practical examples. In Section 3.2, we demonstrate the practical application of strong locality by introducing a technique that we call "password template inversion" for closely-related passwords generation. Finally, we propose a possible attack scenario using strong locality and password template inversion, i.e., SSPG, in Section 3.3.

3.1 Password strong locality

As we briefly introduced in Section 2.1, the latent representation learned by the generator enforces geometric relation among latent points that share semantic relations in the data space. As a consequence, the latent representation maintains *"similar"* instances closer[6] each other in the latent space.

In general, the concept of similarity harnessed in the latent space of a deep generative model solely depends on the modeled data domain (e.g., images, text). In the case of our **passwords latent representation**, this concept of similarity mainly relies on a few key factors such as the structure of the password, the occurrence of common substrings, and the class of characters. Figure 2 (obtained by t-SNE [40]) depicts this observation by showing a 2D representation of small portions around three latent points (corresponding to three sample passwords "`jimmy91`", "`abc123abc`", and "`123456`") in the latent space. Looking at the area with password "`jimmy91`" as the center, we can observe how the surrounding passwords share the same general structure (5L2D i.e., 5 letters followed by 2 digits) and tend to maintain the substring "jimmy" with minor variations. Likewise, the area with the string "`abc123abc`" exhibits a similar phenomenon; where such string was not present in the selected train-set and does not represent a common password template.

As this property of the latent space forces passwords in the vicinity to share very specific characteristics, such as identical substrings, we refer to it as **passwords strong locality.** The passwords strong locality property asserts that latent representation of passwords that share specific characteristics are organized close to each other in the latent space.

One of the direct consequences of the geometric ordering imposed by strong locality is that it provides a natural way to generate a specific class of passwords. Hence, if our aim is to generate passwords strictly related to a chosen prototype password x, then we simply have to fetch latent points around the latent representation z of x (i.e., $x = G(z)$). By strong locality,

[6]For any given metrics; neural-based representations tend to be scale-invariant and typically measured using the *cosine distance*.

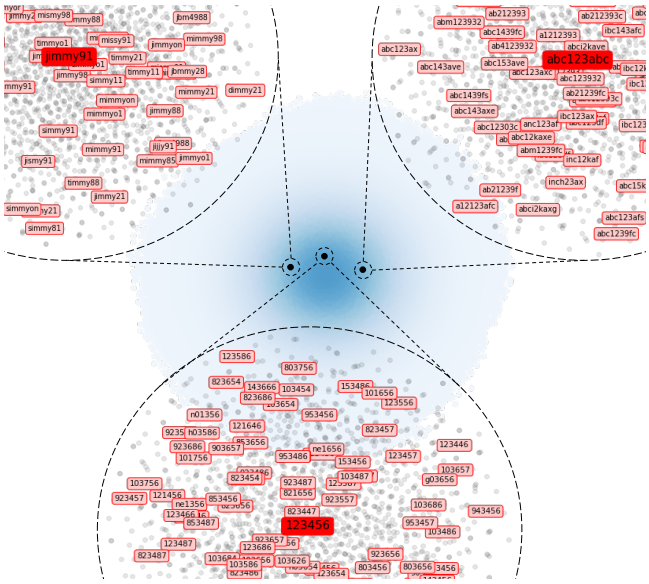


Figure 2: 2D representation of small portions around three latent points corresponding to three sample passwords "`jimmy91`", "`abc123abc`", and "`123456`" in the latent space learned from RockYou train-set. Note: for the sake of better illustration, the image has been cropped.

the obtained latent points should be valid latent representations of passwords with an arbitrary strong relation with x. In this context, we refer the chosen x (or its corresponding latent representation z) with the term **pivot**. The three dark red boxes in Figure 2 are the pivot points in the latent space for their corresponding passwords. However, exploiting this property of the latent space for password guessing requires us to solve **two non-trivial challenges:**

1. For a chosen pivot x, we must efficiently obtain its corresponding latent representation i.e., $z = G^{-1}(x)$.

2. We have to define a technique to explore the latent space surrounding the obtained latent point z.

As described in Section 2.3.2, we solve the first challenge by exploiting an additional network E that is trained to approximate the inverse function G^{-1}. The second challenge of exploring the latent space is solved by restricting the generator to sample from a confined area of the latent space (loosely represented by the small dashed circles in Figure 2). To that purpose, we consider a new latent distribution for the generator. The new distribution has the latent representation of the pivot password as its center and an arbitrary small scale. To remain coherent with prior latent distribution and partially avoid distribution mismatch for the sampled points [58], we chose a Gaussian distribution: $\mathcal{N}(z, \sigma\mathbf{I})$, where the latent representation z of x is obtained through E.

According to the concept of password strong locality, the strength of the semantic relation between a sampled latent point and its pivot should be proportional to the spatial distance between them. As a consequence, the chosen value of σ (i.e., standard deviation) offers us a direct way to control the level of semantic bounding present in the generated passwords. This intuition is better explained by Table 2, where passwords obtained with different values of σ for the same pivot password are reported.

Table 2: The first-ten passwords obtained with different values of σ starting from the pivot string *"jimmy91"*

σ=0.05	σ=0.08	σ=0.10	σ=0.15
jimmy91	jimmy99	mnmm988	jimmy91992
jimmy11	micmy91	tbmmy98	jrm6998
jimmy21	jimsy91	jismyo15	sirsy91
jimmy88	mimmyo1	jizmyon	jrz4988
jimmy81	jbmmy88	j144988	Rimky28
jimmy98	simmy98	jbmm998	missy11
mimmy98	dijmy91	timsy91	jimmy119
jimmy28	jimmy98	jrm4985	sikjy91
simmy91	timsy91	jhmmy88	licky916
mimmy91	jnmm988	jhmm988	gimjyon

Lower values of σ produce highly aligned passwords while larger values of σ permit to explore far from the pivot and to produce a different type of *"similar"* passwords. As shown in Table 2, all the passwords generated with $\sigma = 0.05$ retained not only the structure of the pivot (i.e., 5L2D), but also observed minor variations coherent with underlying password distribution. On another side, passwords generated with $\sigma = 0.15$ tend to escape the password template imposed by the pivot and reaching related-but-dissimilar password structures (e.g., "`jimmy91992`" and "`j144988`").

3.2 Proximity passwords generation with password template inversion

As briefly discussed in Section 3.1, the password strong locality property offers a natural way to generate a very specific/confined class of passwords for a chosen pivot, a task accomplished by exploiting an encoder network E. This encoder is trained to approximate the inverse function G^{-1}, and it is the only tool we have to explore the latent space meaningfully. The default behavior of the encoder is to take as an input a string s and precisely localize the corresponding latent representation in the latent space. As showed in Table 2, sampling from a distribution centered on the obtained latent point, allows us to generate a set of related passwords. However, this approach alone does not find wide possibilities of application on its own in the password guessing scenario.

In this Section, we show that it is possible to "trick" the encoder network to further localize general classes of passwords in the latent space. The users can arbitrarily define these classes via a minimal **template**, which expresses the fine-/coarse-grained definition of the target password. This general approach offers straightforward applications in a real-world scenario. Some of those scenarios will be discussed and empirically evaluated in the next sections.

The encoder network can be forced to work around a specific password definition by introducing a **wildcard** character into its alphabet. The wildcard character - represented with the symbol '$*$' in this paper - can be used as a placeholder to express the presence of an unspecified character. For instance, the template "`jimmy**`" expresses a class of passwords starting with the string "jimmy" followed by two undefined characters. When the encoder inverts this string, the obtained latent point represents the **center** of the cluster of passwords in the latent space with a total length of 8 characters and a prefix "jimmy". Therefore, sampling around this latent point allows us to generate good realizations (according to $p(\mathbf{x})$) of the input template. Column *A* of Table 3 shows a specimen of such obtained passwords for the template "`jimmy**`". In practice, we implement this behavior by mapping a wildcard character to an empty one-hot encoded vector when the matrix corresponding to the input string is given to the encoder. The wildcard characters can be placed at any position or in any quantity to define an arbitrary complex passwords template; some examples are reported in the second row of Table 3.

An intuitive and interesting aspect of this approach is that the wildcards are substituted with the most probable characters following the distribution $p(\mathbf{x})$ (i.e., the probability distribution modeled by the generator). This phenomenon can be observed in the generated samples (Column *A* of Table 3): wildcards in most of the generated passwords have been substituted with digits to potentially reproduce the very common password pattern '*lower_case_string+digits*' [55]. On the contrary, passwords from the template "`*****91`" are reported in Column *E* of Table 3. In this example, we are asking the generator to find 7 characters long passwords with the last two characters as digits. Here, the generated passwords tend to lie towards two most likely password classes for this case, i.e., '*lower_case_string+digits*' and *all_digits*. On the other hand, templates with more observable digits (e.g., Column *F* of Table 3) end-up generating *all_digits* passwords with higher probability.

Template-based passwords generation with state-of-the-art tools: To the best of our knowledge, state-of-the-art tools cannot perform this type of password generation. The probabilistic approaches such as Markov Model (MM)-based and RNN-based (e.g., FLA [42]) are unable to match the expressiveness offered by our wildcard-based approach. In the case of MM, the assumed *n*-Markov property limits the prediction of characters based on substrings of size *n* or less only, but not more. More importantly, the forward-directionality of the process eliminates the possibility of a efficient estimation of wildcards occurring prior to a given substring (e.g., the case in Column *C* of Table 3).The issue of forward-directionality also affects the character level RNN used by FLA, where

the probability of an exponential number of passwords must be computed before using the characters in the template to prune the passwords tree. This is the case of the template reported in column E where the required computational cost for FLA is not very far from computing all the passwords into the chosen probability threshold and filter the ones coherent with the template. On the contrary, the generation of a password **for every template** costs a single network inference[7] following our approach. Additionally, this issue persists even in the case of a bi-directional RNN [51], where the model would fail to efficiently model the "occasionally-punctured" (i.e., plain characters between wildcards) templates such as the ones reported in the Column *D* and Column *F* of Table 3.

The tools that potentially can match the efficiency of our approach are the non-probabilistic ones. In particular, such approaches can take as an input a single string and derive millions of passwords by applying the mangling rules. Usually, the language used to write these rules is very expressive and can match the expressivity of our approach. However, such state-of-the-art tools use human-crafted rules whereas our approach is **totally unsupervised**. Moreover, the mangling rules are fixed before the password generation process and are plainly applied over every dictionary entry. In other words, there is no relation between the input string and the applied rule[8]. Our approach eventually overcomes this limitation; the generated passwords are dependent and univocally induced from the used template. This means that only passwords coherent with the template and the approximate password distribution are eventually generated.

3.3 SubString Password Guessing (SSPG)

An interesting scenario in the domain of password guessing is when the target-password is partially known. There are, at least, two practical situations when the target-password is indeed partially available: (1) targeted-attacks by which an adversary targets a particular user(s) via side-channels [16, 18, 41, 56] and other similar approaches to infer the password of the victim. Such attacks often reveal only part of the password/text correctly because of attacks' accuracy [22]; and (2) when the user forgets her/his own password but remembers it partially, a pretty common situation due to the characteristics of human memory.

Formally, we consider a scenario where an attacker possesses a non-empty set of arbitrary small substrings $S = \{s_0, \dots, s_n\}$ of an unknown target password x. The attacker does not have any knowledge about the correct position of the substring(s) in x as well as about the length of x. But, we consider that the available substring(s) is error-free. The attacker aims at recovering the full password x using the information offered by S. We refer to this scenario by the name of *SubString Password Guessing (SSPG)*. Thanks to the attributes of strong locality, SSPG can also be intuitively performed with erroneous/noisy substrings. We will discuss the performance of SSPG with noisy substrings in Appendix B.

As discussed in Section 3.2, the strong locality property enforces a geometric relation among passwords having common substrings. Consequently, passwords sharing at least one substring are likely to be located in a few specific clusters in latent space. A natural solution to perform a smart SSPG is to localize these clusters and sample latent points from them. By mapping the sampled latent points in the data space, we would indeed be able to generate good candidate passwords that contain the required substrings with high probability. The first obstacle in this procedure arises due to the way these clusters are distributed in the latent space. Passwords of different lengths tend to be organized in different sections of the latent space[9]. The reason for such an organization is that the length of a password is modeled as one of the core explanatory factors [19] by the latent representation. Consequently, passwords with different lengths are distributed far from each other. For instance, the password "`123456`" will be sufficiently far from the password "`123456789`" even if these two passwords share a significantly large substring "123456"[10]. Next,

[7] Two, if we also count the template inversion, which is performed once.

[8] Intuitively, not every mangling rules have equal reason to be applied on all the dictionary entries.

[9] We will exploit this property in Section 4.1 for a different type of password guessing.

[10] In other words, it is unlikely that we can reach to the latent representa-

Table 3: A representative example of exploiting strong locality property over a generator trained on RockYou train-set for samples password templates with $\alpha = 0.15$. Note: results that are coherent with the corresponding template and are in no particular order.

A	B	C	D	E	F	G	H	I
jimmy**	**jimmy****	****jimmy**	****mm*91**	*****91**	**12***91**	**A*****	***A***	**Ra****91**
jimmy6s	jimmy2285	Tujimmy	pumma91	8688691	1228091	ALMYEN	mlqAl25	Rade7891
jimmy65	jimmy8tkm	b3jimmy	dgmmr91	rmuj091	1285j91	AIA655	ROOAGRN	Rani7091
jimmymj	jimmy8yg6	grjimmy	summi91	jukjs91	1204791	A@IN74	AENAYN7	Rajum691
jimmy55	jimmy7t54	lmjimmy	djmmm91	tgk7791	1235691	AIYN66	5ENALL6	Rajemr91
jimmy84	jimmy8576	4jjimmy	ymmmr91	gmbt591	1286891	AqAk62	4ENA128	Raidak91
jimmy20	jimmy5565	egjimmy	mgmms91	tujke91	1256291	ADMSN4	1DqAg45	Rajj8591
jimmyce	jimmy8jj4	djjimmy	timma91	1618591	1250m91	A@ADSk	MOGAEN3	Ranej691
jimmy2j	jimmyka76	jijimmy	djmmy91	mjug791	1205091	A.b474	HANAY49	Rasej891
jimmy14	jimmymj65	jgjimmy	jamma91	tgg7691	1280791	AqIm66	llqArej	Ratul991
jimmy15	jimmy2276	6jjimmy	kumm291	1315891	128ik91	AISr28	JOPA127	Rame6791

a similar hindrance results from the latent representation of a substring position inside the password. For instance, the latent representations of "jimmy91" will be far from the one of "91jimmy".

In other words, passwords containing a given substring are distributed in more than one cluster in the latent space. Therefore, to correctly assess every possible password, we have to cover all such clusters during the password generation process. Accordingly, the challenge is to localize those clusters in the latent space correctly. Nevertheless, we already know an efficient way to localize those clusters; that is, using our template inversion (discussed in Section 3.2). Consider the following example: if we want to localize the clusters of passwords of length 8 that starts with "jimmy", we have to invert the template "jimmy***". Likewise, we have to use "jimmy*****" for passwords of length 10 that starts with "jimmy". Next, the same can be done for the position of the substring, e.g., "**jimmy", "*jimmy*", and "jimmy**" for passwords of length 7 containing "jimmy" somewhere. Therefore, for any substring, we can easily spot all the valid zones of the latent space by enumerating all the possible password templates containing that substring. As a representative example, we obtain all the possible 20 templates (shown in Table A.2 in Appendix A) for passwords with maximum length 10 that contain the substring "jimmy". The same operation can be performed for multiple disjointed substrings just by computing the possible valid templates. The pseudo-code for our SSPG approach is shown in Algorithm 1. The symbols E and G refer to the Encoder and the Generator, respectively whereas the routine *enumerateTemplates* is a function that returns a list of the valid templates for a given set of substrings S and a maximum password length. The operation $x_i \vdash t$ means that the generated password x_i complies with the template t.

Algorithm 1 SubString Password Guessing (SSPG)

Input: Substrings set: S, Int: n, Real: σ
Output: Passwords set: X

$X = \{\}$
$T =$ enumerateTemplates(S)
foreach $t \in T$ **do**
  $z^t = E(t)$
  for i:=1 **to** n **do**
    $z_i \sim \mathcal{N}(z^t, \sigma\mathbf{I})$
    $x_i = G(z_i)$
    if $x_i \vdash t$ **then**
      $X = X \cup \{x_i\}$
    end if
  end for
end for
return X

tions of password "123456789" using string "123456" as pivot.

3.3.1 Evaluation

In this Section, we evaluate our approach against the state-of-the-art mangling rules-based approaches, i.e., JTR [8], HashCat [4], and PCFG [57] for the SSPG scenario. We have already discussed in Section 3.2 that due to their inefficiency, MM-based and RNN-based tools are not a practical choice for this type of attack. We are aware that the state-of-the-art tools are not fully suitable or designed to perform SSPG. F**or a fair comparison, we limit our approach to generate the same number of valid passwords as the most performing competitor tool and focus on the qualitative comparison of the generated passwords.** To further increase the fairness, we choose the most suitable SSPG scenario for the competitor tools: i.e., by considering the substring set $S = \{s_0, \ldots, s_n\}$ containing just one single entry s.

Datasets
We use the LinkedIn password leak [11] as the base dataset for the performance evaluation of different tools. This password leak is composed of over $6 \cdot 10^7$ unique passwords. From this set, we keep passwords with length 10 or less obtaining $4.5 \cdot 10^7$ unique passwords, which is ~ 5 times the RockYou train-set used to train our model (details in Section 2.3.1). For a holistic evaluation, we created two sub-datasets from this filtered LinkedIn dataset. These sub-datasets are created by selecting passwords that contain peculiar substrings. In other words, these substrings model two different level of attacker's knowledge and attack-scenarios:

1. The substrings used to create the first sub-dataset is a list of $\sim 5K$ English first names [3]. The aim of this evaluation set is to model a scenario where an attacker knows the first name of the target. The first names are usually public and commonly used as part of the passwords. As a representative example, the LinkedIn leak has $\sim 12M$ passwords containing English first names [3]. Moreover, the first names are also frequently used as part of common/-classic password templates that can be easily reproduced by mangling rules. To create the sub-dataset, we proceed as follows: for each name s in the list, we take all the passwords X_s in the filtered-LinkedIn dataset that have s as a proper substring in it: i.e., $X_s = \{x \mid \forall x \in \mathbf{X} : s \dot{\in} x\}$. Here, $\dot{\in}$ is the substring operator (i.e., "is a substring in") and $\mathbf{X}$ is the filtered-LinkedIn dataset. Next, we further retain only those passwords in X_s that have a minimum cardinality of 5000 in the filtered-LinkedIn dataset. This filtering gave us the final set $\mathbf{X}_{\text{names}} = \{X_{s_1}, \ldots, X_{s_m}\}$ containing $\sim 4 \cdot 10^6$ passwords and $m = 359$. The average length of these 359 selected substrings is 4.22 and are composed of lower-case characters only.

2. The second sub-dataset represents a more general case, and it also better fulfills the preconditions discussed in Section 3.3. In this case, we select k-most common substrings

with minimum-length of 3 that are present in the passwords of the LinkedIn leak. Similar to the previous sub-dataset of $\sim 5K$ English first names, we select $k = 5000$. However, we select passwords in a set X_s with minimum cardinality of 100000 in the filtered-LinkedIn dataset as the average length of selected substrings is 3.02. The final sub-dataset $\mathbf{X}_{\text{lss}}$ is composed of $\sim 3 \cdot 10^7$ passwords and $m = 237$. A small sample of the selected substrings shown in Table 4.

Table 4: A small sample of the selected substrings used in the second sub-dataset

199	202	777	987	a19	a20	ate	ati
cat	era	eri	her	ink	man	min	oma
ree	res	ria	san	sta	1234	2011	love

Evaluation setup

Given an evaluation set (i.e., $\mathbf{X}_{\text{names}}$ or $\mathbf{X}_{\text{lss}}$), we evaluate each tool in the task of password guessing over each subset of passwords (i.e., X_s) separately. In the process, every tool is exposed to the substring s, which is exploited for generating guesses. Then, the intersection between the set of generated password/guesses and X_s is computed, i.e., the number of generated passwords that match the passwords in X_s. Finally, the overall cardinality of the matched passwords for the entire evaluation set (i.e., $\mathbf{X}_{\text{names}}$ or $\mathbf{X}_{\text{lss}}$) against the total number of generated passwords for the entire evaluation set (i.e., $\mathbf{X}_{\text{names}}$ or $\mathbf{X}_{\text{lss}}$) is used as the evaluation criterium for each tool.

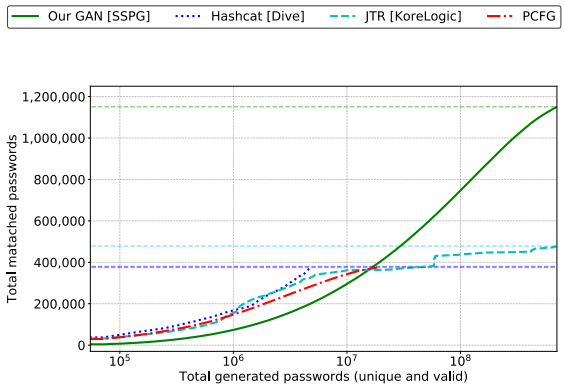
We generate guesses by using the Algorithm 1 for our approach. In particular, we use $S = \{s\}$ and $\sigma = 0.35$ (chosen empirically) in the algorithm for each X_s. Next, the value of n has been chosen to match the maximum number of passwords produced by one of the state-of-the-art tools.

For mangling rules-based tools (i.e., HashCat and JTR), we implement SSPG by keeping the substring s as the only entry in the dictionaries and applying the rules on this single-entry-dictionary to generate passwords. For each of these two tools, we chose the largest set of rules available (to the best of our knowledge). These rules are KoreLogic [10] for JTR and Dive [2] for HashCat, which are respectively composed of over $5K$ and $99K$ rules. We generate every possible password using the rule-sets and only consider those passwords as valid passwords that contain the substring s as a proper substring.

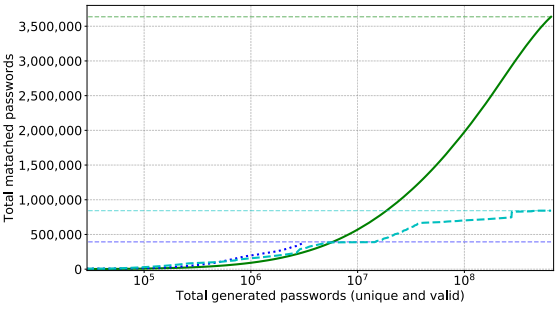
For PCFG, we train/learn its grammar by using the same dataset (i.e., RockYou train-set) used to train our generator, and by using the default parameters [13]. During the password generation process, we use s as the only valid entry for the *Alpha* variables of PCFG [57] whereas the entries for other types of variable (e.g., *Capitalization* and *Digits*) are kept unaltered. We can evaluate PCFG only with $\mathbf{X}_{\text{names}}$ because it can not model variables that represent strings composed of mixed character classes (such as "a19" and "a20" reported in Table 4 for $\mathbf{X}_{\text{lss}}$). Here, we generate 10^7 passwords with each s and retain only the unique passwords containing the given substring s.

Results

Figure 3 (a) and Figure 3 (b) show the results for different tools over the evaluation sets $\mathbf{X}_{\text{names}}$ and $\mathbf{X}_{\text{lss}}$, respectively. As mentioned in Section 3.3.1, each line depicts the sum of the matched passwords against the total number of generated valid passwords for the entire evaluation set. HashCat, JTR, and PCFG (when applicable) generate a heterogeneous number of valid passwords. For HashCat and JTR, that value is directly influenced by the number and types of word-mangling rules. JTR with KoreLogic rule-set produces the highest number ($\sim 10^9$) of valid passwords. Therefore, we tune the variable n in our Algorithm 1 to produce passwords in the same magnitude. At the bottom, PCFG is able to produce just $\sim 2 \cdot 10^7$ valid passwords despite the higher limits of requested passwords. The reason for such a low number of passwords is the small size of grammars produced after the training, whose number cannot be increased.



(a) $\mathbf{X}_{names}$



(b) $\mathbf{X}_{lss}$

Figure 3: Matched passwords against generated valid passwords by different tools over evaluation sets $\mathbf{X}_{\text{names}}$ and $\mathbf{X}_{\text{lss}}$

In case of $\mathbf{X}_{names}$, HashCat generates the least number of valid passwords. PCFG generates more passwords than HashCat but matches slightly fewer passwords than HashCat. On the other side, JTR generates the highest number of valid

passwords and matches more passwords than both HashCat and PCFG. In this experiment, our GAN model matches over 240% more passwords than JTR for equal number of guesses. It is important to highlight that in contrast to other tools, our GAN model can continue to generate (depending on n in Algorithm 1) and efficiently match more passwords.

In case of $\mathbf{X}_{lss}$, HashCat again generates the least number of valid passwords whereas JTR again generates the highest number of valid passwords. Nevertheless, both these tools match a significantly smaller number of passwords as compared to our GAN model in SSPG mode. Our approach matches over 900% and 400% more passwords than HashCat and JTR, respectively. The reason why both HashCat and JTR perform poorly is that the mangling rule-sets are designed to match most probable passwords and are not intended to work with random strings. Hence, in such a complex situation, our approach - that is not biased to a specific scenario - can perform significantly better than the other state-of-the-art tools.

For the interested readers, Figure A.1 in Appendix A shows distribution of matched passwords over each substring for both evaluation sets $\mathbf{X}_{\text{names}}$ and $\mathbf{X}_{\text{lss}}$.

4 Passwords weak locality and Dynamic Password Guessing (DPG)

In this Section, we present our second major contribution, i.e., the password weak locality concept and its possible applications in the field of password guessing. In Section 4.1, we introduce the concept of password weak locality with the help of different practical examples. Section 4.2 presents DPG from theoretical (Section 4.2.1) as well as practical (Section 4.2.2) point of view. Finally, we mention potential applications of DPG.

4.1 Password weak locality

The embedding properties of the latent representation map passwords with similar characteristics close to each other in the latent space. We called this property strong locality, and we exploited it intending to generate variations of a chosen pivot password or template (discussed in Section 3.1). In that case, the adjective "strong" highlights the strict semantic relation among the generated set of passwords. However, the same dynamics that enables the strong locality also allows a more generic and broad form of semantic bounding among passwords. This latter property seems to be able to partially capture the general features of the whole passwords distribution. Such features could be very abstract properties of the distribution, such as the average passwords length and character distribution due to password policies. We refer to this observed property as **password weak locality** in contrast with the strong locality.

As a representative example, Figure 4 depicts the 2D representation of passwords from *myspace* [12], *hotmail* [7], and *phpbb* [14] on the latent space learned by a generator on the RockYou train-set[11]. We can observe that the passwords coming from the same (within one) dataset tend to be concentrated in the latent space and do not spread abruptly all over the spectrum. This can be traced back to the fact that passwords sharing very general features (e.g., like those coming from the same passwords distribution) are mapped close to each other in wide but bounded zones of the latent space.

The dimensionality of the fraction of latent space covered by an entire passwords set (the red parts in Figures 4 (a), (b), and (c) clearly depends on the heterogeneity of its passwords. Passwords from smaller sets (e.g., *myspace*) are concentrated in restricted and dense zone of the latent space, whereas passwords from larger sets (e.g., as *phpbb*) tend to cover a bigger section while they are still closely knitted.

In the following sections, we will present evidence of the strong locality property, and we will show how to exploit this property of the latent space to improve password guessing.

4.2 DPG for covariate shift reduction

First, we present the theoretical motivation behind DPG in Section 4.2.1 followed by its possible implementation in Section 4.2.2.

4.2.1 Theoretical motivations

Probabilistic password guessing tools implicitly or explicitly attempt to capture the data distribution behind a set of observed passwords, i.e., the train-set. This modeled distribution is then used to generate new and coherent guesses during a password guessing attack. A train-set is usually composed of passwords that were previously leaked. By assumption, every password-set leak is characterized by a specific password distribution $p^*(\mathbf{x})$. When we train the probabilistic model, we implicitly assume $p^*(\mathbf{x})$ to be general enough to well-represent the entire class of password distributions. This generality is essentially due to the fact that the real-word password guessing attacks are indeed performed over sets of passwords that potentially come from completely different password distributions. As a matter of fact, we typically do not have any information about the distribution of the attacked-set. This can indeed be completely different from the one used for model training. As a representative example, different password policies or users' predominant languages can cause the test-set's distribution to drastically differ from the train-set's distribution. This discrepancy in the distribution of the train-set and test-set is a well-known issue in the domain of machine learning, and it is referred to as *covariate shift* [53].

[11] It is important to emphasize that these graphical depictions are obtained by a dimension reduction algorithm. Hence, they do not depict latent space accurately. So, they merely serve as a representative illustration. We will verify our assumption empirically later in the paper.

As stated above, typically, we do not know anything about the distribution of the attacked-set. However, once we crack the first password, we can start to observe and model the attacked distribution. Every new successful-guess provides valuable information that we can leverage to improve the quality of the attack, i.e., to reduce the *covariate shift*. This iterative procedure recalls a Bayesian-like approach since there is continuous feedback between observations and probability distribution. However, we highlight that in our case we do not use neither a prior nor a posterior probability distribution.

For fully data-driven approaches - a naive solution to incorporate such new information - is to fine-tune the model so as to change the learned password distribution. However, *prescribed probabilistic models* such as FLA directly estimate the password distribution using a parametric function:

$$p(\mathbf{x}) = p(\mathbf{x};\theta), \tag{5}$$

where θ is the set of weights of a neural network. In this case, the only chance to modify the distribution $p(\mathbf{x})$ in a meaningful way is to act on θ harnessing a learning process. However, this is not an easy/attractive solution. Mainly because the new guessed passwords are potentially not enough representative[12] to force the model to generalize over the new information. Additionally, the computational cost of applying fine-tuning on the network is considerable, and sound results cannot be ensured due to the sensitivity of the process.

Similar to FLA, our generative model also exploits a neural network as an estimator. However, the modeled distribution is a joint probability distribution, shown in Eq. 6:

$$p(\mathbf{x}) = p(\mathbf{x},\mathbf{z}) = p(\mathbf{x};\theta)p(\mathbf{z}), \tag{6}$$

where $p(\mathbf{z})$ is referred to as the latent distribution.

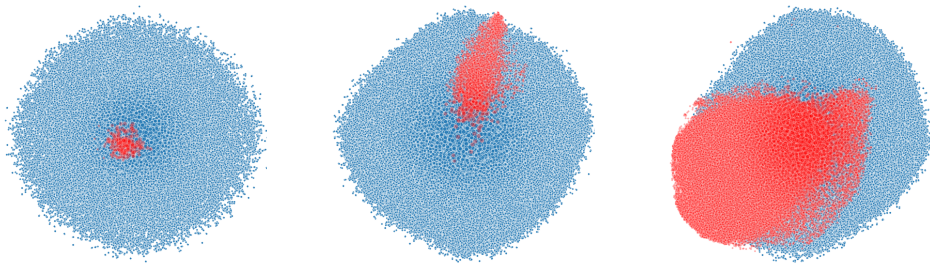
As introduced in Section 2.1, when $p(\mathbf{z}) = \dot{p}(\mathbf{z})$ (i.e., prior latent distribution), $p(\mathbf{x};\ \theta)p(\mathbf{z})$ is provable to be a good approximation of the target data distribution (i.e., the distribution followed by the train-set). Nevertheless, $p(\mathbf{z})$ can be arbitrarily chosen and used to indirectly change the probability distribution modeled by the generator. The RHS of the Eq. 6 clearly shows that θ is not the only free parameter affecting the final passwords distribution. Indeed, $p(\mathbf{z})$ is completely independent of the generator, and so can be modified arbitrarily without acting on the parameters of the neural network.

This possibility, along with the passwords weak locality of the latent space, allows us to correctly and efficiently generalize over the new guessed passwords, leading the pre-trained network to model a password distribution closer to the attacked one. It is noteworthy that this capability of generalizing over the new points is achieved via the weak locality and not from the neural network itself. **The intuition here is that when we change $p(\mathbf{z})$ to assign more density to a specific guessed password x, we are also increasing the probability of its neighboring passwords that, due to the weak locality property, are the passwords with similar characteristics.** This, in turn, makes possible to highlight the general features of the guessed passwords (e.g., structure, length, character set, etc.), instead of focusing on its more fine-grained[13] and specific aspects.

So, by controlling the latent distribution, we can choose to increase the probabilities of the zones that are potentially covered by the passwords coming from the attacked distribution. We call this technique **Dynamic Password Guessing (DPG)**. In the case of homogeneous distribution (e.g., *myspace*), we can narrow down the solution space around the dense zones, and avoid exploring the whole latent-space. On the other side, for passwords sets sampled from distributions far from the one modeled by the generator, we can focus on zones of the latent space which, otherwise, would have been poorly explored. In both cases, we can reduce the *covariate shift* and improve the performance of the password guessing attack.

[12] A very few cracked passwords against a dataset of millions of uncracked passwords.

[13] Features that do not give us hints on the attacked password distribution.



(a) *myspace* (b) *hotmail* (c) *phpbb*

Figure 4: Password Weak Locality: 2D visualization of the latent points for three different passwords sets for a generator trained on the RockYou train-set. The red points represent the latent points corresponding to the passwords in the respective password set whereas the blue points loosely represent the dense part of the latent space. Please refer to the color version for better illustration.

4.2.2 Practical implementation

In this Section, we explain DPG from a practical point of view. Algorithm 2 briefly describes the DPG.

Algorithm 2 DPG to reduce *covariate shift*

Input: Set: O, Int: α

```
i = 0
p_latent = ṗ(x)
Z = {}
foreach z ~ p_latent do
    x = G(z)
    if x ∈ O then
        i++
        Z_i = Z = Z ∪ {z}
        if i ≥ α then
            p_latent = makeLatentDistribution(Z_i)
        end if
    end if
end for
```

Here, O represents the target set of passwords, Z is the collection of all the passwords guessed by the generator, and α is defined as the **hot-start** parameter of the attack, an ingredient that we describe later in this section. The variable p_{latent} in the pseudo-code, represents the latent distribution from which we sample latent points. For simplicity, we use the notation $z \sim p_{\text{latent}}$ to directly express the sampling operation of a latent point z from the latent space according to the distribution p_{latent}. The procedure *makeLatentDistribution* returns the latent distribution induced from the group of guessed passwords Z_i at step i. Leveraging the maximum-likelihood framework, we choose such distribution to maximize the probability of the set of observed passwords $X_i = \{G(z) \mid z \in Z_i\}$ according to Eq. 7 by using the latent distribution $p(\mathbf{z})$ as the only free parameter.

$$p(X_i) = \int p(X_i;\theta)p(\mathbf{z})\, dz. \tag{7}$$

This is accomplished by considering a latent distribution $p(\mathbf{z} \mid Z_i)$ conditioned to the set of passwords guessed at each step i. The final password distribution represented by the generator during the DPG is reported in Eq. 8.

$$p(\mathbf{x}) = p(\mathbf{x};\theta)p(\mathbf{z} \mid Z_i). \tag{8}$$

As a natural extension of the proximity password generation harnessed in Section 3.2, we choose to represent $p(\mathbf{z} \mid Z_i)$ as a finite mixture of isotropic Gaussians. In particular, the mixture is composed by n Gaussians, where: (1) n is the number of the latent points in Z_i; and (2) for each $z_j \in Z_i$, a Gaussian is defined as $\mathcal{N}(z_j, \sigma\mathbf{I})$ with center as z_j and a fixed standard deviation σ.

When the probability of a password, i.e., $x_j = G(z_j)$, is known[14], we weight the importance of the j^{th} distribution as $P(x_j)$; otherwise a uniform distribution among the Gaussians is assumed. Equation 9 defines the probability density function of the latent space.

$$p(\mathbf{z} \mid Z_i) = \sum_{j=0}^{n} P(G(z_j)) \cdot \mathcal{N}(\mathbf{z} \mid z_j, \sigma\mathbf{I}). \tag{9}$$

Every new guessed password x introduces a new Gaussian centered at z to the mixture. As a consequence, every new guessed password contributes to changes in the latent distribution $p(\mathbf{z} \mid Z_i)$ by moving the density of the distribution in the zone of the latent space where it lies. Figure 5 visualizes this phenomenon.

Figure 6 depicts the performance comparison between a static attack (e.g., PassGAN) and the DPG over the three passwords sets. Adaptively changing the latent distribution allows us to boost the number of guessed passwords per unit of time. In the *phpbb* set, we match $\sim 5\%$ additional passwords with respect to the static attack technique. Importantly, this improvement comes without any additional information or assumption over the attacked passwords set. In addition, the computational overhead due to the new sampling technique is negligible. The steep improvement in the performance obtained with the DPG gives additional support to our assumption made on the weak locality of the latent space. Furthermore, it confirms that reducing the *covariate shift* has a direct and concrete impact on the number of guessed passwords.

The sudden growth in the guessed passwords in the DPG (shown in Figure 6) is due to the hot-start or α parameter. In other words, we use the prior latent distribution until a predetermined number (α) of passwords have been guessed. After that, we start to use the conditional latent distribution $p(\mathbf{z} \mid Z_i)$. The reason is that if the DPG starts with the very first guessed password, then the latent distribution can be stuck in a small area of the latent space. However, launching DPG after guessing a sufficient number of passwords (i.e., after finding a set of unbiased latent points in the latent space) gives us the possibility to match a heterogeneous set of passwords, which correctly localize the dense zones of the latent space where the attacked passwords are likely to lie. These observations are also evident with our empirical results shown in Figure 7, which depicts a comparison among the static attack, a DPG with $\alpha = 15\%$, and a DPG with $\alpha = 0\%$ (i.e., no hot-start). These results confirm that the absence of hot-start indeed affects and eventually degrades the performance of the DPG.

[14] We known its frequency in the attacked set of passwords. In a off-line attack, this is usually the case.

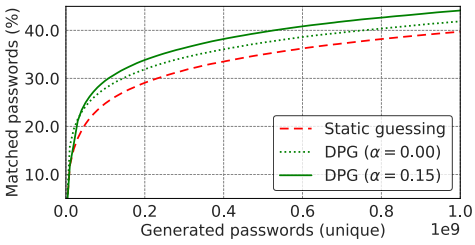


Figure 7: The impact of α on the performance of DPG for *phpbb* test-set

The final hyper-parameter of our attack is the standard deviation (σ) assigned to every Gaussian in the mixture. This value defines how far we want to sample from the clusters of observed passwords. A larger value of σ allows us to be less biased and explore a wider zone around the guessed passwords; whereas a smaller value permits a more focused inspection of the latter. Therefore, the value of σ can be interpreted as the parameter controlling the **exploration-exploitation** trade-off[15] in the attack. Figure 8 depicts the effect of different values of σ on the performance of DPG. Smaller values of α yields better overall results. This outcome suggests that it is not necessary to sample too far from the dense zones imposed by Z_i, and rather a focused exploration of those zones is beneficial. This observation is perfectly coherent with the concept of weak locality, giving further support to the speculated ability of the latent space of capturing and translating general features of an entire password distribution in geometric relations.

[15] A trade-off often occurring in reinforcement learning.

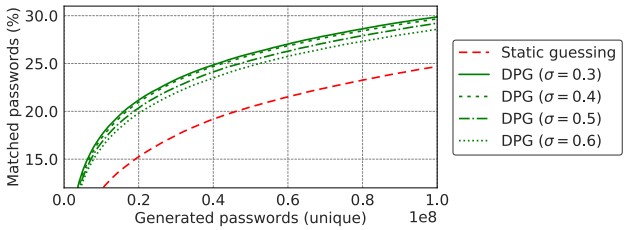
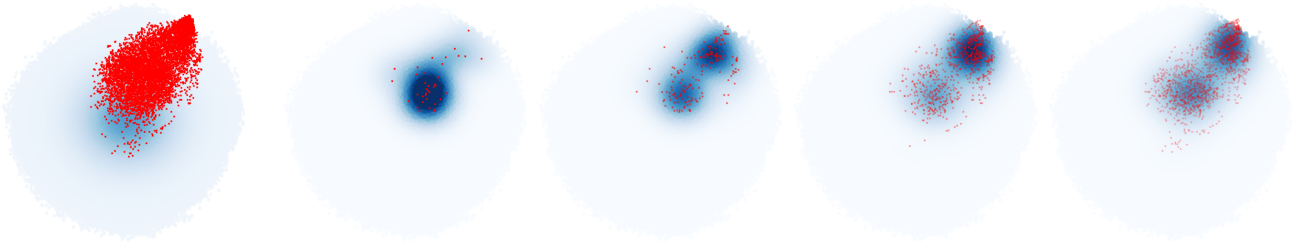


Figure 8: The impact of σ on the performance of DPG for *phpbb* test-set

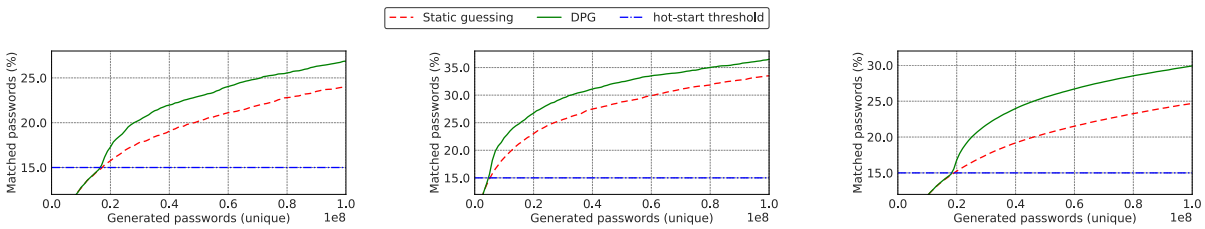
Applications and conclusions: We demonstrated that the DPG framework offers a direct way to deal with the *covariate shift* phenomenon that naturally occurs in real-world password guessing scenarios. Furthermore, the potential of DPG is not limited to this specific phenomenon. Following are a few potential applications of DPG:

- DPG can be easily extended to support a form of "bootstrapping" for the password guessing attack. As an example, consider a situation in which the attacker knows a small set



(a) Actual attacked set (b) 10^4 generation (c) 10^5 generation (d) 10^6 generation (e) 10^7 generation

Figure 5: 2D visualization of: (a) the entire *hotmail* dataset (red-part) mapped on the latent space learned from the RockYou train-set and (b-e) the latent space in four progressive attack steps for DPG on the *hotmail* test-set. The red markers portray the guessed passwords at each step (i.e., the Z_i), whereas the color intensity of the blue regions depicts the probability assigned from the used latent distribution (i.e., mixture of Gaussians) to the latent space. Please refer to the color version for better illustration.



(a) *myspace* (b) *hotmail* (c) *phpbb*

Figure 6: The performance gain obtained by DPG (with α = 0.15) with respect to static attack for three different test-sets

of passwords from the targeted password set. The attacker can utilize this additional information to create a bias in the password guessing attack and boost its performance even before the attack. More precisely, these known passwords can be modeled in the latent distribution of the generator by moving the density around the new latent points that are obtained with the encoder network. In practice, this can be easily achieved by initializing the set Z (Algorithm 2) with these known passwords. Moreover, these latent points could also be obtained using the template inversion technique presented in Section 3.2. This is the case in which the attacker retrieves, somehow, only partial information and not the entire passwords.

- Hitaj et al. demonstrated [35] that GAN-based models are able to produce a class of passwords that is significantly different (disjoint) from the passwords obtained using other state-of-the-art tools. Therefore, combining such different password guessing approaches allows us to improve the final outcome [35]. Our DPG technique offers a direct way to enhance the performance of such combinations further. As a matter of fact, DPG allows us to focus on the zones of latent space that were not covered (due to its design/bias) by the first tool in the pipeline. Consequently, DPG technique can significantly increase the probability of generating guesses that were not generated by the previous tool.

In conclusion, the building blocks of the DPG, i.e., the malleable latent density and dynamic approximation of the attacked password distribution, are generalized approaches that can be used to increase the attack's performance under various assumptions. These concepts can be easily extended and are naturally open to various applications.

5 Related Works

Password guessing is a classical attack, by which an attacker tries to guess the right password by repeatedly testing various candidate passwords. Systematic studies on password guessing dates back to 1979 [44], and probably, password guessing attacks exist since the inception of the concept of passwords [20]. Since a vast number of works have been proposed in this active area of research, we limit the discussion to state-of-the art tools and techniques used for password cracking in this section.

Narayanan et al. [45] proposed to use standard Markov modeling techniques from natural language processing to generate password guesses. Their approach requires manual intervention for defining password rules that describe the structure of the generated passwords. Weir et al. [57] extended this technique via Probabilistic Context-Free Grammars (PCFGs). In particular, Weir et al. [57] showed a technique to "learn" the password rules from a given set of passwords. Durmuth et al. [28] and Ma et al. [39] have also proposed enhancements in this direction of password guessing.

John The Ripper (JTR) [8] and HashCat [4] are the two most widely used password guessing tools. Both JTR and HashCat have demonstrated their effectiveness at guessing/recovering the passwords from several leaked password dataset [1]. Both the tools support a number of password guessing strategies including: (1) classical brute-force attacks; (2) dictionary-based attacks; (3) rule-based (also called mangled wordlist) attacks [6, 10], which is one of the most exploited technique; and (4) Markov model-based attacks [5, 9].

Ciaramella et al. [23] introduced neural networks for password guessing in their seminal work. In the same line of development, Melicher et al. [42] proposed FLA (Fast, Lean, and Accurate) that uses recurrent neural networks [32, 54] to estimate the password distribution, which is then used to guess the strength of a password. Hitaj et al. [35] presented PassGAN that uses a GAN to autonomously learn the distribution of real passwords from actual password leaks, and to generate password guesses.

Similarly to our SSPG framework, different works have focused on creating password variations for a given starting password [24, 46], primarily with the aim of modeling credential tweaking attacks. Credential tweaking is a targeted attack where the adversary knows the targeted user's credentials for one or more services and aims to compromise accounts of the same user on other services. Different from credential stuffing, here user's passwords are suppose to be "tweaked" versions[16] of the known ones. In this direction, Pal [46] et al. proposed novel attack/defense techniques for credential tweaking. Both the attack and the defense techniques are built on top of a password similarity concept. They model a specific form of semantic similarity by using a supervised dataset of user-password pairs. They assume the *distributional hypothesis* for passwords to be **true**, and define two passwords to be 'similar' if they are often chosen together by users. The proposed attack technique is founded on a probabilistic neural model, and it aims to produce tweaked variations of an input password. The produced variations are then used as suitable guesses for the targeted tweaking attack. More interestingly, their defensive technique firstly glances the application of supervised representation learning in password guessing. Their technique is based on constructing an embedding space that is learned using the out-of-the-shelf word embedding tool. In such space, the geometric relation between passwords is used to estimate the similarity between chosen passwords. This similarity measure is then used to build a "Personalized Password Strength Meter" that aims to spot the use of tweaked password by the user at password creation time. In contrast to our password representation, their embedding space does not allow sampling operation and so passwords generation as well.

[16] The user can create such password variations to accommodate passwords composition policies of different services.

6 Conclusion and future works

Orthogonal to the current research directions, we propose a complete paradigm shift in the task of password guessing. We demonstrate that locality principles imposed by the latent representation of a GAN generator can open new practical and theoretical possibilities in this field. Based on these properties, we propose two new password guessing frameworks, i.e., SSPG and DPG. SSPG, along with its underlying foundation, i.e., the template-based password generation, is useful in several real-world scenarios. We empirically demonstrated its efficiency over its potential competitors. DPG demonstrates that the knowledge from freshly guessed passwords can be successfully generalized and used to reduce the *covariate shift* phenomenon. We believe that these properties can also be used to do an efficient estimation of password guessability. We will explore this possibility in our future efforts.

Availability

The code, pre-trained models, validation sets, and other materials related to our work are publicly available at: https://tinyurl.com/yyqbv7n2

References

[1] "Cracking Passwords 101". https://tinyurl.com/y268xahe.

[2] "Dive-Rules". https://tinyurl.com/yxm9t6ov.

[3] "English First Names". https://tinyurl.com/y2tz8asq.

[4] "hashcat". https://tinyurl.com/y636jsz9.

[5] "HashCat Per Position Markov Chains". https://tinyurl.com/y2l3vggx.

[6] "HashCat Rules". https://tinyurl.com/y55h6s9j.

[7] "Hotmail Password Leak". https://tinyurl.com/yyr2je4m.

[8] "John the Ripper". https://tinyurl.com/j91l.

[9] "John the Ripper John the Ripper - Markov Generator". https://tinyurl.com/7gd9xt4.

[10] "KoreLogic-Rules". https://tinyurl.com/yxdrftpc.

[11] "LinkedIn Password Leak". https://tinyurl.com/yxf7f5gv.

[12] "MySpace Password Leak". https://tinyurl.com/y433aaah.

[13] "PCFG GitHub". https://tinyurl.com/yyhavsld.

[14] "phpbb Password Leak". https://tinyurl.com/yxonf7um.

[15] "RockYou Password Leak". https://tinyurl.com/af858jc.

[16] Kamran Ali et al. Keystroke Recognition Using WiFi Signals. In *ACM MobiCom*, pages 90–102, 2015.

[17] Martin Arjovsky et al. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[18] Davide Balzarotti et al. Clearshot: Eavesdropping on Keyboard Input from Video. In *IEEE S&P*, pages 170–183, 2008.

[19] Yoshua Bengio et al. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[20] Hossein Bidgoli. *Handbook of Information Security, Information Warfare, Social, Legal, and International Issues and Security Foundations*, volume 2. John Wiley & Sons, 2006.

[21] Andrew Brock et al. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[22] Shuo Chen et al. Side-channel Leaks in Web Applications: A Reality Today, A Challenge Tomorrow. In *IEEE S&P*, pages 191–206, 2010.

[23] Angelo Ciaramella et al. Neural Network Techniques for Proactive Password Checking. *IEEE Transactions on Dependable and Secure Computing*, 3(4):327–339, 2006.

[24] Anupam Das et al. The Tangled Web of Password Reuse. In *NDSS Symposium*, pages 1–15, 2014.

[25] Peter J Diggle and Richard J Gratton. Monte Carlo Methods of Inference for Implicit Statistical Models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):193–212, 1984.

[26] Jeff Donahue et al. Adversarial Feature Learning. *arXiv preprint arXiv:1605.09782*, 2016.

[27] Vincent Dumoulin et al. Adversarially Learned Inference. *arXiv preprint arXiv:1606.00704*, 2016.

[28] Markus Dürmuth et al. Omen: Faster Password Guessing using an Ordered Markov Enumerator. In *ESSoS*, pages 119–132, 2015.

[29] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv preprint arXiv:1701.00160*, 2016.

[30] Ian Goodfellow et al. Generative Adversarial Nets. In *NIPS*, pages 2672–2680, 2014.

[31] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *Elsevier Knowledge-Based Systems*, 151:78–94, 2018.

[32] Alex Graves. Generating Sequences with Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850*, 2013.

[33] Ishaan Gulrajani et al. Improved Training of Wasserstein GANs. In *NIPS*, pages 5767–5777, 2017.

[34] Kaiming He et al. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016.

[35] Briland Hitaj et al. PassGAN: A Deep Learning Approach for Password Guessing. In *ACNS*, pages 217–237, 2019.

[36] Christian Ledig et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *CVPR*, pages 4681–4690, 2017.

[37] Yang Li and Tao Yang. Word Embedding for Understanding Natural Language: A Survey. In *Springer Guide to Big Data Applications*, pages 83–104. 2018.

[38] Junyu Luo et al. Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets. In *International Conference on Neural Information Processing*, pages 207–216. Springer, 2017.

[39] Jerry Ma et al. A Study Of Probabilistic Password Models. In *IEEE S&P*, pages 689–704, 2014.

[40] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[41] Philip Marquardt et al. (sp)iPhone: Decoding Vibrations From Nearby Keyboards Using Mobile Phone Accelerometers. In *ACM CCS*, pages 551–562, 2011.

[42] William Melicher et al. Fast, Lean, and Accurate: Modeling Password Guessability using Neural Networks. In *USENIX Security Symposium*, pages 175–191, 2016.

[43] Shakir Mohamed and Balaji Lakshminarayanan. Learning in Implicit Generative Models. *arXiv preprint arXiv:1610.03483*, 2016.

[44] Robert Morris and Ken Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11):594–597, 1979.

[45] Arvind Narayanan and Vitaly Shmatikov. Fast Dictionary Attacks on Passwords using Time-space Tradeoff. In *ACM CCS*, pages 364–372, 2005.

[46] Bijeeta Pal et al. Beyond Credential Stuffing: Password Similarity Models using Neural Networks. In *IEEE S&P*, pages 1–18, 2019.

[47] Alec Radford et al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv preprint arXiv:1511.06434*, 2015.

[48] Tim Salimans et al. Improved Techniques for Training GANs. In *NIPS*, pages 2234–2242, 2016.

[49] Pouya Samangouei et al. Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. *arXiv preprint arXiv:1805.06605*, 2018.

[50] Thomas Schlegl et al. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. In *Springer International Conference on Information Processing in Medical Imaging*, pages 146–157, 2017.

[51] Mike Schuster and Kuldip K Paliwal. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[52] Casper Kaae Sønderby et al. Amortised MAP Inference for Image Super-resolution. *arXiv preprint arXiv:1610.04490*, 2016.

[53] Masashi Sugiyama et al. Covariate Shift Adaptation by Importance Weighted Cross Validation. *Journal of Machine Learning Research*, 8(May):985–1005, 2007.

[54] Ilya Sutskever et al. Generating Text with Recurrent Neural Networks. In *ICML*, pages 1017–1024, 2011.

[55] Blase Ur et al. I Added '!'at the End to Make It Secure: Observing Password Creation in the Lab. In *SOUPS*, pages 123–140, 2015.

[56] Martin Vuagnoux and Sylvain Pasini. Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. In *USENIX Security Symposium*, pages 1–16, 2009.

[57] Matt Weir et al. Password Cracking using Probabilistic Context-free Grammars. In *IEEE S&P*, pages 391–405, 2009.

[58] Tom White. Sampling Generative Networks. *arXiv preprint arXiv:1609.04468*, 2016.

[59] Roman V Yampolskiy. Analyzing User Password Selection Behavior for Reduction of Password Space. In *ICCST*, pages 109–115, 2006.

[60] Jun-Yan Zhu et al. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *IEEE ICCV*, pages 2223–2232, 2017.

Appendix A Supplementary tables & figures

Table A.1 lists the value of hyper-parameters used to train the encoder network. Table A.2 shows the templates for passwords with maximum length of 10 that contain a substring "jimmy". Figure A.1 shows the distribution of matched passwords over each substring for our approach against the most performing competitor tool in the task of error-free SSPG.

Table A.1: Hyper-parameters used to train our encoder network

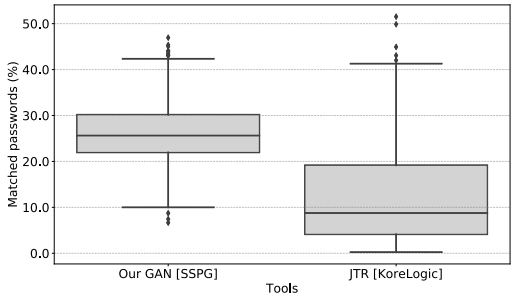
Hyper-parameter	**Value**
α	0.2
β	0.2
γ	0.6
Batch size	64
Learning rate	0.001
Optimizer	*Adam*
Temperature decay step	250000
Temperature limit	0.1
Temperature scheduler	*polynomial*
Train iteration	$3 \cdot 10^5$

Table A.2: Templates for passwords with maximum length of 10 that contain a substring "jimmy"

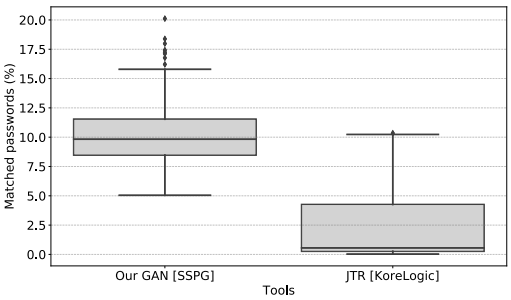
*****jimmy	****jimmy*	***jimmy**	**jimmy***	*jimmy****
jimmy*****	****jimmy	***jimmy*	**jimmy**	*jimmy***
jimmy****	***jimmy	**jimmy*	*jimmy**	jimmy***
**jimmy	*jimmy*	jimmy**	*jimmy	jimmy*

Appendix B SSPG with noisy substring

SSPG comes handy naturally in scenarios where partial information - a substring - of the target-password is known. SSPG assumes that the known substring is correct. But, what if the known substring is not completely accurate? In other words, how SSPG performs if the the substring is erroneous (e.g., "min" instead of "man")? This situation can arise due to the following reasons for the two cases that we mentioned in Section 3.3: (1) inaccuracy of the side-channel attack; and (2) incorrect remembrance of the password by the user. Formally, we now consider an attack scenario where the available substring s' is a noisy version of the true substring s that is actually present in the target password x.



(a) $\mathbf{X}_{names}$



(b) $\mathbf{X}_{lss}$

Figure A.1: The distribution of matched passwords over each substring for our approach against the most performing competitor tool using evaluation sets $\mathbf{X}_{\text{names}}$ and $\mathbf{X}_{\text{lss}}$

The password strong locality property offers us a way to deal with this kind of scenarios as well. As shown in Section 3.1, it allows us to explore passwords with common substrings as well as the passwords with variations of the substrings. The samples reported in Table 2, present some such cases. Single character variations e.g., 's' in "simmy91" or 'm' in "mimmy91", are reachable from "jimmy91" with a lower value of σ (i.e., 0.05). On the other side, reaching higher-character variations requires a higher value of σ, e.g., "sirsy91" from "jimmy91" needs $\sigma = 0.15$.

As shown in our results (Section 3.3.1), the state-of-the-art tools do not work at all or, at most, perform inefficiently SSPG with error-free substring (Section 3.3) whereas none of the state-of-the-art tools is designed to model SSPG with noisy substring. To be specific, enumerating passwords containing s' with character level RNN [42] does not provide any information about the passwords containing s. Likewise, applying word mangling rules on s' is unlikely to produce passwords with s as substring. On the contrary, our approach with strong locality will map passwords containing s' or s closer in the latent space, given that s' and s are similar to each other (e.g., "man" and "min" against "abc" and "9$t"). Therefore, sampling around the pivots induced by s' can also cover the passwords related to s depending on the similarity of s' and s as well as chosen value of σ. Furthermore, the chosen value of σ can be used to reflect the confidence that

an attacker has in the eavesdropped s'. The only modification required in our Algorithm 1 to perform this SSPG with noisy substring is to remove the if clause at line number 8.

To give empirical support to our claim, we repeat the SSPG experiment with $\mathbf{X}_{\text{lss}}$ (Section 3.3.1), but, now with a noisy version of the substrings. In particular, we compute the noisy substring s' for each s (in other words, for each X_s) by applying a distortion function on s. This distortion function selects a random character in s and substitutes it with another random character of the same character-class. Then, we apply the SSPG algorithm (Algorithm 1) with $S = \{s'\}$ and the same value of n. We chose two different values of σ: 0.35 and 0.50. The passwords obtained from these new experiments are evaluated against the results with the exact substring, i.e., we take matched-passwords with s as the ground truth instead of the entire set X_s. Figure B.1 depicts the proportion of matched passwords with respect to the experiment with exact substrings for the two new experiments.

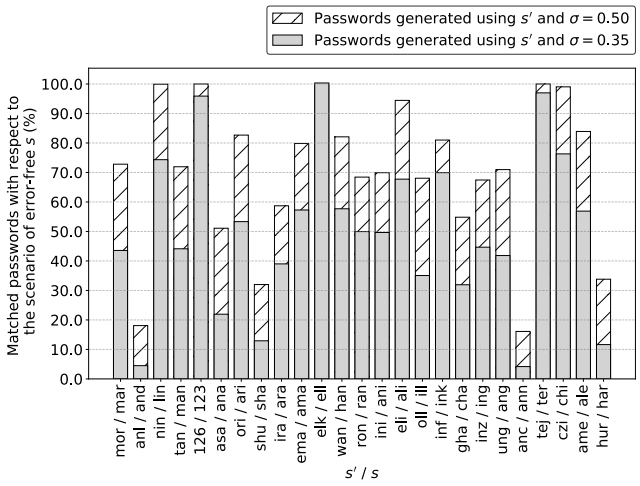


Figure B.1: Effect of σ on SSPG with noisy substring s' against SSPG with error-free s

To present a clear picture, we report only 25 random entries. With $\sigma = 0.35$ (the same value used for the experiment with the exact substring), we are able to cover part of the previous results. Hence, for the dynamics mentioned above, we can match passwords even if the substring used to localize the zones of the latent space is partially erroneous. However, the obtained results are heterogeneous; in few cases we match high number of passwords while we match less in others. We believe that this is intrinsically related to the organization of the latent space where not all perturbations are treated equally. Nevertheless, increasing the value of σ enables us to explore far from the pivots induced by the noisy string, which increases the probability to cover the areas of latent space dedicated to the clean string s. As evident in our results, increasing the value of σ uniformly improves the overall performance.