On the Security of Cracking-Resistant Password Vaults

Maximilian Golla
Horst Görtz Institute
Ruhr-University Bochum
Bochum, Germany
maximilian.golla@rub.de

Benedict Beuscher
Horst Görtz Institute
Ruhr-University Bochum
Bochum, Germany
benedict.beuscher@rub.de

Markus Dürmuth
Horst Görtz Institute
Ruhr-University Bochum
Bochum, Germany
markus.duermuth@rub.de

ABSTRACT

Password vaults are used to store login credentials, usually encrypted by a master password, relieving the user from memorizing a large number of complex passwords. To manage accounts on multiple devices, vaults are often stored at an online service, which substantially increases the risk of leaking the (encrypted) vault. To protect the master password against guessing attacks, previous work has introduced *cracking-resistant password vaults* based on Honey Encryption. If decryption is attempted with a wrong master password, they output plausible-looking decoy vaults, thus seemingly disabling offline guessing attacks.

In this work, we propose attacks against cracking-resistant password vaults that are able to distinguish between real and decoy vaults with high accuracy and thus circumvent the offered protection. These attacks are based on differences in the generated distribution of passwords, which are measured using Kullback–Leibler divergence. Our attack is able to rank the correct vault into the 1.3 % most likely vaults (on median), compared to 37.8 % of the best-reported attack in previous work. (Note that smaller ranks are better, and 50 % is achievable by random guessing.) We demonstrate that this attack is, to a certain extent, a fundamental problem with all *static* Natural Language Encoders (NLE), where the distribution of decoy vaults is fixed. We propose the notion of *adaptive* NLEs and demonstrate that they substantially limit the effectiveness of such attacks. We give one example of an adaptive NLE based on Markov models and show that the attack is only able to rank the decoy vaults with a median rank of 35.1 %.

Keywords

Password Managers; Natural Language Encoders; Honey Encryption; Cracking-Resistance

1. INTRODUCTION

Passwords are still the de-facto standard for online user authentication, despite substantial drawbacks in terms of memorability and security. To relieve the user from the burden of memorizing a large number of passwords (ideally not shared between services, hard to guess, and not related to the service name or personal memories), many IT security professionals recommend the use of password vaults (also called password managers) [25]. Password vaults store passwords and usually also domains and usernames, in an encrypted container, where the encryption key is derived from a master password using a key derivation function (KDF) [18]. Vaults can store both user-chosen passwords, which can be chosen stronger as the user does not have to memorize them, and randomly chosen cryptographically strong passwords.

To facilitate migrating the stored passwords to a new or secondary device, and to backup data to prevent loss, many vaults offer the possibility to store the encrypted vault with an online service. Password vaults stored online are a promising target for attackers. This is not a theoretical problem: *LastPass* was a target of a suspected breach in 2011 and 2015 [15]. Recent work, analyzing the security of online password managers [20, 29], has shown a number of weaknesses, including vulnerabilities that allow exfiltration of stored vaults.

An attacker can try to recover the missing master password, once the encrypted vault has been stolen [13, 19, 5]. In an offline guessing attack, the attacker can try candidate (master) passwords, trial-decrypt the vault, and then verify if the candidate was correct by observing the result of the decryption. Often decryption with an incorrect master password will yield an error or a malformed header, allowing easy identification of wrong candidates. This kind of attack is "offline", as no interaction with an online service is required and the correctness of a guess can be verified locally. The number of password guesses an attacker can try is almost unbounded, only limited by the computational resources at disposal. No data is publicly available describing how users choose their master passwords. Utilizing the available information for normal account passwords [4, 11] and given the recent advancements in GPU and FPGA design [35] we postulate that also user-chosen master passwords can be guessed in limited time [14, 30]. For current vaults [1, 27], the number of guesses per day on a single GPU is in the order of 10^9 to 10^{12} guesses. Thus, unthrottled guessing attacks would constitute a major threat. In principle, this problem can be solved by avoiding giving feedback about the successful decryption. However, not only explicit feedback in the form of error messages needs to be avoided, but also implicit feedback, i. e., in the form of "implausi-

CCS '16, October 24–28, 2016, Vienna, Austria.

ISBN 978-1-4503-4139-4/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978416

ble" passwords in the vault. One solution was devised in the *Kamouflage* scheme [3], which constructs so-called decoy vaults that are generated during a setup phase, which are encrypted under *similarly structured* decoy master passwords, and return predefined plausible password vaults. An attacker will get multiple vaults in an offline guessing attack and needs to decide, e. g., via online verification, which vault is the correct one. A newer proposal, *NoCrack* [9], improved on this approach by not only offering a predefined (constant) number of decoy vaults but by generating new and plausible decoy vaults on the fly for each (wrong) master password candidate. Additionally, they discovered a flaw in the generation of Kamouflage's decoy master passwords that led to an improved attack against the scheme. Generating reasonable looking decoy vaults is not a trivial task. NoCrack uses techniques from Honey Encryption (HE) and Natural Language Encoders (NLEs) based on Probabilistic Context-Free Grammars (PCFGs) to generate plausible decoy vaults. PCFGs have been shown to quite accurately model password distributions in the past [34]. In a preliminary evaluation, the authors showed that basic machine learning attacks are not able to distinguish real from decoy vaults.

Contribution.

The features used in the security analysis of the NoCrack vaults were quite simplistic: repeat counts, edit distances, and n-gram frequency statistics were used as input to the machine learning step. We show that techniques exist that can distinguish real from decoy vaults with high accuracy.

Our technique is based on the distribution of the passwords in the vaults, which can be easily measured by an attacker simply by trial-decrypting a vault with a number of wrong master passwords. By determining the similarity between distributions (we use Kullback–Leibler divergence as a measure of similarity) one can see that the distribution of passwords in the decoy vaults, generated by NoCrack, is substantially different from the distribution of other password lists. This enables us to rank the correct vault significantly higher (up to a median rank of 1.97 % for real-world vaults and 0.1 % for artificial vaults composed from real-world password lists). We show that, to some extent, this is not a problem unique to NoCrack, but also caused by differences in various password lists. Based on the observation that this problem persists for many different ways to choose decoy vaults, we propose the notion of *adaptive NLEs*, where the generated distribution of decoy vaults depends on the actual values stored in the vault.

Finally, we evaluate additional signals that enable one to even better distinguish real from decoy vaults. We show that additional information, such as the correlation of usernames and passwords or password policies, should be considered by the NLE. In the case of NoCrack, this results in a mean rank of 2.4 % and a 1st Quartile of 0.56 %, an improvement by a factor of 40 and 170, respectively. To summarize, our contributions include:

(i) We show that techniques exist that can distinguish real from decoy vaults with high accuracy for NoCrack.
(ii) We propose the notion of *adaptive NLEs*, where the generated distribution of decoy vaults depends on the actual values stored in the vault.
(iii) We evaluate signals that enable one to even better distinguish real from decoy vaults via additional information such as usernames and password policies.

Overview.

In Section 2 we will review some material about cracking-resistant password vaults and introduce the attacker model that we consider. In Section 3 we will define the concept of adaptive NLEs, as opposed to static NLEs. In Section 4 we will show that for static NLEs and specifically for NoCrack, KL divergence is able to distinguish between real and decoy vaults with high accuracy. In Section 5 we will describe some more factors that can be used to distinguish real from decoy vaults. In Section 6 we will define an adaptive NLE based on Markov models and demonstrate its much stronger resistance against attacks compared to static NLEs.

2. CRACKING-RESISTANT VAULTS

In the following, we introduce the required notions and review some material about cracking-resistant password vaults and describe the attacker model that we consider.

Recall that an offline guessing attack describes an attack where one can verify guesses without contacting an online service. Thus, the number of tests that can be performed is only limited by the available computational resources. In contrast, an online service can implement rate-limiting and risk-based authentication to limit the exposure in online attack scenarios.

2.1 Kamouflage

Bojinov et al. [3] have proposed a solution for this problem, by designing a password vault that resists offline guessing attacks. Their Kamouflage system uses a "hide-in-an-explicit-list" approach. For this, they generate a large number (they suggest 10^7 for medium security) of decoy vaults that are stored besides the real vault. Even by correctly guessing a master password, the attacker no longer knows whether decrypting the vault with the master password leads to the real or one of the decoy vaults present in the file.

To generate plausible looking decoys, they were required to consider that multiple passwords are generated for the same user. Therefore, they implemented a solution that generates decoys by assigning probabilities to password templates that are derived by a process similar to the concept of using a PCFG. They tokenize every given domain password and reuse those tokens across the passwords in the vault, where a token represents a word or number of a certain length. Subsequently, they validate the tokens using a dictionary and flag unknown tokens for manual user review. Based on those derived tokens, they generate plausible looking decoys via a dictionary.

A potential drawback of this approach is the storage overhead, required to save a large number of decoy vaults. Chatterjee et al. [9] broke the scheme by abusing the revealed structure of the master password, once any (real or decoy) vault master password has been guessed. By identifying the tokens (e. g., "Kamouflage16" $\rightarrow L_{10}D_2$) of the found password, one is able to narrow down the search space significantly and speed up the search for the remaining master passwords. This flaw even degrades the resistance against offline guessing attacks to a lower level than traditionally encrypted vaults that do not use decoys at all.

2.2 Honey Encryption

Honey Encryption (HE), introduced by Jules and Ristenpart [17], produces a ciphertext that when decrypted with any incorrect key, yields plausible-looking decoy plaintexts,

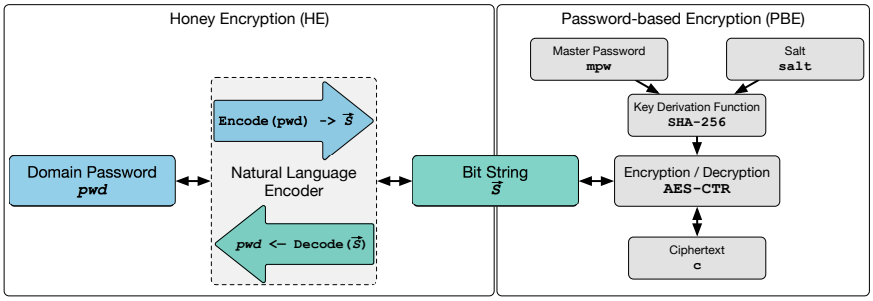

Figure 1: Design of NoCrack (simplified). This schematic omits details, e. g., domain hashing or separated handling of human-chosen and computer-generated passwords. The Natural Language Encoder (NLE) decodes a bit string to a password and encodes vice versa.

called honey messages. It addresses the problem of encrypting plaintext using low-entropy keys, such as keys derived from passwords, by applying a specialized encoding mechanism first and encrypting the result afterward. The key challenge of building such system is the construction of a randomized message encoding scheme, called distribution transforming encoder (DTE). Creating such an encoder is relatively easy for some classes of data, like a random string, but very challenging for natural language text like real-world passwords. The authors showed the usefulness of Honey Encryption for password-based encryption schemes by building a system for encrypting RSA secret keys and credit card numbers. Utilizing an HE-scheme for encrypting a password vault provides, in contrast to Kamouflage's solution, security beyond the traditional offline work bound. In other words, it is ensured that an attack is never less expensive than attacking a traditional vault.

2.3 NoCrack and Natural Language Encoder

Chatterjee et al. [9] used the idea of Honey Encryption to provide more secure password vaults, extending the concept previously applied in Kamouflage. An overview of the simplified architecture of NoCrack is given in Figure 1. Basically, the idea is to output "plausible looking" password vaults for *each* master password used for trial decrypting a vault (whereas Kamouflage only outputs decoy vaults for a small number of wrong master passwords). Here, the challenge is to generate plausible-looking vaults on the fly, which is achieved by building a new kind of DTE for encoding and decoding of natural language named Natural Language Encoder (NLE). For this, they built a DTE that takes natural language, i. e., a password *pwd*, as input and encodes it to a random bit string $\vec{S}$. Reverse, decoding any random bit string $\vec{S}$ outputs a plausible plaintext, i. e., a password *pwd*. A number of promising candidates for NLEs are available (and often used for password guessing [34, 11, 6] or password strength meters [7] already): Probabilistic Context-Free Grammars (PCFG), n-gram Markov models, password samplers, and more.

NoCrack's Approach: A PCFG Model-based NLE.

NoCrack implements two different DTEs called *UNIF* (uniform) for randomly chosen passwords and *SG* (sub-grammar) for human-chosen passwords. While the former is straightforward to build, the construction of the latter is an unsolved problem and a challenging task. The authors of NoCrack sketched two different approaches for this, an n-gram model and a PCFG-based solution and described the latter in detail. PCFGs can be used to model a distribution of strings in a language, i. e., passwords, by associating a probability for the members of the distribution defined by an underlying context-free grammar (CFG). In language theory, a CFG is a formal grammar that is defined by a lexicographically ordered set of production rules. A rule is specified as a pair (A, α), where A is a non-terminal and α is a terminal or non-terminal symbol.

One can build a PCFG by the assignment of probabilities to such CFG rules. Due to the need to be able to encode every occurring password, there is a special *catch-all* rule with a very low probability. For a given PCFG, a password may be specified by a sequence of probabilities. Based on the first probability in the sequence, a rule from the set is selected for the start symbol S producing the children of S in a parse tree. Via recursion, the complete parse tree is built. They showed that a rule set probability can be represented as an integer. By this, given a PCFG, one can represent a password by a non-unique vector of integers.

In the encoding step of the NoCrack NLE, the vector of probabilities that define a parse tree, is uniformly selected from all such producing the given password. During decoding the given vector of probabilities, thus the parse tree, is used to rebuild the encoded password.

The authors proposed an SG (sub-grammar) NLE that is used to build a vault from the described single password NLE (MPW). Applying a single password NLE multiple times produces a vault of unrelated passwords that is rather insecure. Instead, the SG approach tries to build related passwords, to simulate normal user behavior more accurately. For this, all passwords stored in the real vault are parsed using the trained PCFG model. After this, a new sub-grammar PCFG that consists of the cumulative set of rules used during the parsing is built. Finally, the rule probabilities are copied from the original PCFG and renormalized over the sub-grammar PCFG. Please note that some special rules, like the aforementioned *catch-all* rule is always part of the used grammar. To be able to decode the SG encoded passwords, the SG is encoded as well and always the first part of the NLE's output. For full details, we refer the interested reader to the original paper [9].

The NoCrack Implementation.

The authors of NoCrack implemented a full version of a Honey Encryption-based password vault. The implementation must be considered a prototype which does not always

correctly decrypt a vault and occasionally crashes. However, we did not focus on the manager software itself, like the implemented client–server model and encryption, but rather on the concept of the actual PCFG model-based NLE. While the software [8] is still evolving, we used the version as of September 24th, 2015 for our experiments.

NoCrack, as implemented currently, does not store usernames, so technically we have to assume that usernames are known. Usernames are typically not considered secret, so we expect them to be easily retrievable, with the email address, or it may be even publicly known.

However, we would expect that any practical implementation would also store the usernames in plaintext to be able to fill out forms automatically.

2.4 Attacker Model

While cracking-resistant vaults, of course, are subject to all the normal attacks that can be launched against vaults, here we concentrate on so-called *online-verification attacks*, where an attacker performs online queries to verify which vault (resulting from an offline guessing attack) is the correct one.

1. *Trial-Decryption:* The attacker trial-decrypts the given encoded vault with N master password candidates according to a password distribution that is believed to be suitable for the task. This yields a number of candidate vaults $cv_1, \ldots, cv_N$.
2. *Ranking of Vault Candidates:* The attacker ranks the vault candidates such that the "more likely correct" vaults are (hopefully) ranked near the top. The original paper uses machine learning to rank the more likely candidate vaults to the top.
3. *Online Verification:* Finally, the attacker uses online guessing to verify the correctness of the vaults, starting with the highly ranked vaults. The number of online verification attempts the attacker is to perform depends on the countermeasures implemented by the service and can vary wildly. Note that even a single service with very weak defenses has a great impact on the security of the complete vault.

The security of a vault scheme against this type of attack is best measured by the distribution of ranks of the real vault. To this end, it is not necessary to create millions of decoy passwords; as those are chosen uniformly by the NLE anyway it is good enough to observe the ranking of the real vault in a much smaller set of decoy vaults. The average rank of the real vault among the decoy vaults is a good measure for the average defense against online-verification attacks.

2.5 Further Related Work

Besides cracking-resistance, other security and usability aspects of password vaults were studied. Especially, the security of web-based password managers was examined by Li et al. [20]. They found flaws in the function for generating one-time passwords, the use of bookmarklets, and a password sharing feature. An attack that abuses the autofill functionality was described by Silver et al. [29]. They described multiple ways to execute their so-called *Sweep Attack*, by injecting JavaScript on the fly into the victim's browser. The potential threat by cross-site scripting (XSS) attacks was analyzed by Stock et al. [31]. McCarney et al. [22] investigated the usability of a dual-possession-based vault alternative. Their solution leverages a desktop computer and a smartphone to build a theft-resistance system without the use of a master password. Further, Gasti and Rasmussen [13] analyzed the formal security of encrypted vault storage formats. They provided two realistic security models and evaluated a number of popular password managers.

3. STATIC AND ADAPTIVE NLES

A central aspect of an NLE used in a password vault is the distribution of its generated decoy vaults. It needs to generate decoy vaults that cannot be easily distinguished from the real vault. Technically, for a traditional vault software, this distribution exists as well, with two vaults, the "error vault" $\perp$ and the correct vault, having a non-zero probability. For Kamouflage this distribution has a limited number of vaults with non-zero probability. To decrypt the vault, the NLE construction, as used in NoCrack, gets a bit string as input. This bit string is generated by applying a KDF to the used master password. Thus, the input distribution for the NLE is (close to) a uniform distribution, and (practically) independent of the distribution of the guessed master passwords. We distinguish two variants of NLEs:

(i) *Static NLEs*: an NLE where the generated distribution of decoy vaults is independent of the actual values stored in the vault.

(ii) *Adaptive NLEs*: an NLE where the generated distribution of decoy vaults depends on the actual values stored in the vault.

Of course, the decoy vaults always depend on the master password.

3.1 Static Distribution NLEs

NoCrack and all NLEs that follow the schematics in Figure 1 are necessarily static, as no information about the stored vault is available to the NLE. Thus, the distribution of decoy vaults is necessarily independent of the passwords stored in the vault.

Static NLEs seem to be a logical and conservative choice for password vaults. In fact, an attacker can easily approximate the generated distribution and if the distribution transports information about the passwords stored in the vault, an attacker might be able to extract information about the vault from this easily accessible distribution.

However, as we will show in the following section, static NLEs have one major drawback and are susceptible to online-verification attacks. In brief, the problem is that the distribution of decoy vaults needs to be fixed at one point before the actual passwords are stored. But password distributions differ substantially from one service to another, where reasons include a different password policy (which may vary over time), a different user-base, different (perceived) security requirements, and much more. A previously fixed distribution will not be able to handle these vast differences that even change over time. Furthermore, storing a strong password in such a vault will make breaking the vault easier; a counter-intuitive behavior that we should avoid at all cost.

3.2 Adaptive Distribution NLEs

A potential solution to the problem described in Section 3.1 is offered by adaptive NLEs, where the chosen distribution of decoy vaults depends on the passwords stored in the vault. This makes it unnecessary to "predict" the changes in password distributions over time at creation time of the software, as the distribution can adapt to relevant changes from

the stored passwords. Obviously, this raises another security concern: When the distribution depends on the stored passwords, will knowledge of the distribution help in recovering the stored passwords? In Section 6 we will show an adaptive NLE based on Markov models, which resists online-verification attacks much better than a static NLE, and additionally, the adaptive property does not undermine the security of the scheme.

4. THE (IN-)SECURITY OF STATIC NLES

Our first main contribution shows that static NLEs and especially NoCrack suffer from a severe weakness that substantially limits their security. In brief, we show how an attacker can efficiently distinguish the distribution of passwords generated by NoCrack from real vaults, and we argue that this is at least partially a fundamental limitation of static NLEs.

4.1 Distinguishing Real from Decoy Vaults

In the attack scenario described in Section 2.4 and introduced by Chatterjee et al., the adversary has created a list of candidate vaults by decrypting the encrypted vault with a number of candidate master passwords. We can assume that the correct master password is in the list of candidate passwords and thus the correct vault is among the list of candidate vaults. Then an attacker wants to rank the available candidate vaults so that the average position of the true vault in the list of candidate vaults is near the top of the list. A "perfect" NLE would lead to an average rank of 50 %, as there was no way to distinguish real from decoy vaults and thus the ranking cannot do better than guessing. Chatterjee et al. tested attacks based on generic ML algorithms. We devise an alternative attack targeting the similarity of the observed distributions, based on the Kullback–Leibler divergence.

Kullback–Leibler Divergence.

The Kullback–Leibler divergence (KL divergence) is a measure of the difference between two probability distributions. Given two probability distributions P and Q, the KL divergence is defined as

$$D_{KL}(P \parallel Q) = \sum_{z \in \mathrm{supp}(P)} P[z] \cdot log \frac{P[z]}{Q[z]},$$

provided that $\mathrm{supp}(P) \subset \mathrm{supp}(Q)$, and ∞ otherwise. We use logarithms to base 2 throughout this work. The measure is not symmetric in P and Q.

Setup.

The setup follows the attack model described in Section 2.4. We highlight the deviations from there in the following.

1. *Determining Distribution P_{decoy} of Decoy Vaults:* Static NLEs have the (defining) property that the distribution of decoy vaults is constant. This distribution can be obtained in two ways: Either, it can be approximated by repeatedly sampling passwords from the distribution by evaluating the KDF and trial-decrypting the vault (similar, but less computationally expensive, one can choose and decode a bit string from uniform random). We use this method in the current section and we determine the influence of the sample size on the accuracy of the attack in Section 4.3. Or, for some NLEs it is possible to use a theoretical argument to derive a mathematical description for the distribution via the source code. We use this method in Section 6.1 for Markov model-based NLEs.
2. *Trial-Decryption:* The attacker trial-decrypts the given encoded vault with master password candidates, yielding candidate vaults $cv_1, \ldots, cv_N$.
3. *Ranking of Vault Candidates:* For the ranking in this experiment we use the similarity of distributions of passwords measured by the KL divergence.
   We compute the similarity scores
   $$s_i := D_{KL}(\hat{P}_{cv_i} \parallel \hat{P}_{decoy}) \qquad \text{for } i = 1, \ldots, N$$
   for each candidate vault cv_i. Here, $\hat{P}_{cv_i}$ is the distribution derived from the vault cv_i based on relative frequencies and $\hat{P}_{decoy}$ is the distribution derived from the empirical measurements in the first step, again based on relative frequencies. We rank the candidate vaults based on the score s_i, where higher s_i means a larger distance from the decoy distribution and thus likely the real vault.
4. *Online Verification:* Finally, the attacker uses online guessing to verify the correctness of the vaults, starting with the higher ranked vaults.

4.2 Datasets

For completeness, we give a brief description of the datasets used in the following evaluations.

The *PBVault* leak is likely from before June 2011 and has a substantial overlap with credentials that were obtained via keystroke logging and HTML form injection by the *ZEUS Trojan*. Chatterjee et al. used this list to evaluate their NLE approach and made the file available along with the NoCrack source code. The file contains username and password pairs. To the best of our knowledge, PBVault is the only publicly available list of password vaults. Detailed statistics on the file, called Pastebin, can be found in the NoCrack paper [9].

The *RockYou* list contains 32 million plaintext passwords. It was leaked in December 2009 through an SQL injection attack. The *Gmail* list contains over 5 million Google Mail credentials and was posted in September 2014 in a Russian Bitcoin forum. The *Yahoo* leak is from July 2012 and was obtained by attacking the Yahoo! Voices publishing service; it contains around 450 thousand passwords. The *MySpace* list contains around 55,000 passwords that were posted in October 2006. The passwords in the list were obtained via a phishing attack.

4.3 Experiments for Entire Vaults: NoCrack SG vs. PBVault

First, we present results using the KL divergence on full vaults. To keep the results comparable with previous work, we use the same set of password vaults used in the evaluation of NoCrack.

Setup.

This experiment follows the description in Section 4.1.

1. The decoy distribution $\hat{P}_{decoy}$ is approximated by the relative frequencies using 30 million samples of entire vaults from the NoCrack distribution, obtained by repeatedly decrypting a vault using a wrong master password and querying it for the passwords for 50 well-known login domains.

Table 1: Rank results based on a KL divergence attack of entire vaults, where smaller numbers mean a more efficient attack. Decoy vaults are chosen from the NoCrack distribution. Real vaults are chosen from the PBVault distribution. For better comparability to previous work [9], we list results for varying classes of vault sizes (left), and results for varying numbers of samples for the reference distribution (right).

KL Div.: NoCrack vs. PBVault			
NoCrack Sample Size: 30×10^6			
	Mean	$Q_{0.25}$	Median
Vault Size 2-3	9.56 %	0.86 %	2.08 %
Vault Size 4-8	5.97 %	0.97 %	1.86 %
Vault Size 9-50	3.14 %	1.12 %	1.69 %
All (2-50)	**6.20 %**	**1.02 %**	**1.97 %**

KL Div.: NoCrack vs. PBVault			
Vault Size: 2-50, By Sample Size			
	Mean	$Q_{0.25}$	Median
100,000	12.83 %	3.99 %	7.99 %
300,000	10.39 %	3.18 %	6.54 %
1,000,000	8.48 %	2.27 %	4.46 %
3,000,000	7.39 %	1.75 %	3.36 %
10,000,000	6.59 %	1.32 %	2.63 %
30,000,000	6.20 %	1.02 %	1.97 %

2. As data for the ranking we use 1,000 vaults, where one "real vault" is chosen from PBVault and 999 "decoy vaults" are chosen from a list of decoy vaults, obtained by repeatedly decrypting a NoCrack vault with the wrong master password and querying it for the passwords for a specific number of login domains. Decoy vaults are chosen to have the same size as the real vault. Note that this list is disjoint from the list used for approximating $\hat{P}_{decoy}$.
3. The ranking is performed using KL divergence $D_{KL}(\hat{P}_{cv_i} \parallel \hat{P}_{decoy})$.
4. This experiment is repeated 100 times for each vault in PBVault, choosing fresh decoy vaults in each iteration.

Results.

The results of this experiment are summarized in Table 1. They show that the real vault cv_{real} is ranked on average among the 6.20 % of the most likely vaults, thus reducing the amount of online guessing by approximately a factor of 16. This is significantly better than the best reported attack by Chatterjee et al. using machine learning [9], which reported an average rank for the combined feature vector of 39.70 %. The situation is, however, even worse when we take a closer look at how the ranks of the real vaults are distributed. In fact, this distribution is skewed to the right, with a median of only 1.97 % and a 1st Quartile of 1.02 %. This means that half the vaults are among the top-ranked 1.97 % of the vaults, reducing the amount of online guessing by a factor of 50, and 25 % of the vaults are ranked among the top 1.02 %, reducing the amount of online guessing by a factor of 98. Note that we provide an attack against NoCrack taking into account further information in Section 5.4, which brings down the average rank to 2.43 % and the 1st Quartile to 0.56 %. For comparability, we also report the results separated by vault sizes, see Table 1 (left). The results vary little with the vault sizes, mainly the mean gets smaller for larger vault sizes, most likely as there is more data available for the comparison.

Influence of the Size of the Training Set.

In order to determine the influence of the number of samples for approximating the decoy distribution, we ran the experiment with varying numbers of samples. The results are summarized in Table 1 (right). As expected, more samples provide better results, while the improvements are getting less pronounced beyond 10 million samples. Note that this depends on the NLE used and we have found different behavior for the NLE based on Markov models, which we introduce in Section 6.

Furthermore, while a KDF, which is computationally expensive to evaluate, will slow down the trial decryptions, it cannot effectively slow down sampling of the decoy distribution. Instead, the NLE can be queried directly by providing a random bit string S as input for the NLE (cf. Figure 1).

4.4 Experiments for Single Passwords: NoCrack MPW vs. Password Leaks

The previous experiments have demonstrated that there is a significant difference in the distribution of passwords sampled from NoCrack and from the PBVault leak and that the KL divergence is a suitable tool to distinguish the two. We strive to better understand how (dis-)similar password lists are, and how well the KL divergence can perform to accurately distinguish them.

All available password lists we are aware of, with the exception of the PBVault, contain single passwords only. Therefore, we sample one single random password from a number of NoCrack (SG) vaults to obtain independent password samples. This approach produces artificial vaults, containing not related but independent passwords, like NoCrack (MPW), that can be meaningfully compared with other artificial vaults built from *single-passwords-only* password lists.

Setup.

In this experiment we perform a pairwise comparison between two password lists. We denote one list as "real password list" R, the other as "decoy password list" D.

1. The decoy list D is split into two (disjoint) parts D_v and D_{ref}, where D_v is used for picking decoy vaults, and D_{ref} is used as an approximation for the reference distribution. As some lists (e. g., MySpace) are quite small, we repeatedly split D for each new vault.
2. We pick one "real" vault of size 10 from the set R, where we select the individual passwords independently of each other. We pick 999 "decoy" vaults of size 10 from D_v, again with independent passwords.
3. When computing the KL divergence, we use the set D_{ref} to obtain the reference distribution (which is disjoint from D_v).

Table 2: Rank results based on KL divergence of artificial (independently selected) vaults of size 10. Note, for easier comparison we also report numbers for the static Markov NLE, which is introduced in Section 6.

All vs. All: Average Rank									
Leak D	RockYou			Gmail			Yahoo		
Leak R	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
RockYou	50.54 %	25.03 %	50.85 %	50.70 %	25.53 %	50.35 %	49.20 %	23.02 %	49.85 %
Gmail	28.04 %	7.01 %	20.12 %	50.16 %	25.53 %	50.15 %	40.61 %	16.42 %	36.94 %
Yahoo	27.54 %	7.01 %	20.62 %	40.05 %	15.92 %	36.04 %	50.38 %	25.40 %	50.80 %
MySpace	27.46 %	6.91 %	19.82 %	28.33 %	8.21 %	21.02 %	29.07 %	9.01 %	22.62 %

All vs. All: Average Rank									
Leak D	MySpace			NoCrack (MPW)			Static Markov (MPW)		
Leak R	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
RockYou	10.53 %	1.90 %	4.00 %	0.18 %	0.10 %	0.10 %	44.49 %	15.82 %	41.44 %
Gmail	8.32 %	1.60 %	3.20 %	0.12 %	0.10 %	0.10 %	8.28 %	0.10 %	0.10 %
Yahoo	7.45 %	1.50 %	3.10 %	0.11 %	0.10 %	0.10 %	5.32 %	0.10 %	0.10 %
MySpace	50.98 %	26.80 %	51.65 %	0.11 %	0.10 %	0.10 %	11.93 %	0.10 %	0.60 %

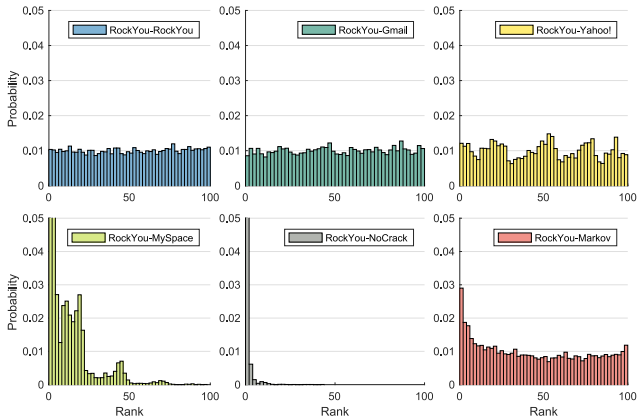

Figure 2: Result distribution of the KL divergence experiment for single (unrelated) passwords. The real vault is sampled from *RockYou*, the decoy vaults are sampled from distribution approximations of real-world passwords and artificial ones, i. e., *NoCrack (MPW)* and *static Markov (MPW)*.

Results.

The results are summarized in Table 2. First, we see that NoCrack performs worse on these artificial vaults, i. e., KL divergence is substantially better at distinguishing the distribution generated by NoCrack from these artificial vaults. As we used independent passwords for NoCrack as well, the differences that allow us to distinguish are caused by the distribution of passwords. The most likely reason for this behavior is that 10 independently chosen passwords carry more information than 10 passwords with a high reuse, as it is observed in the PBVault leak.

Second, we see that the distributions of RockYou, Yahoo, and Gmail are rather hard to distinguish. Apparently, their distributions are relatively similar. Third, MySpace is quite different from RockYou, Yahoo, and Gmail, and is relatively easy to distinguish from them. Lastly, as expected, comparing one distribution against the same distribution has an average rank of 50 %. Note that these results are not symmetric, i. e., it makes a difference which distributions the decoys are chosen from.

5. CRACKING NOCRACK

We have seen a first criterion (the KL divergence) for distinguishing between vaults drawn according to the decoy distribution. Next, we will consider several more criteria that are based on structural differences of the vault.

5.1 Correlation and Dependence

As already mentioned, but not examined by Bojinov et al. [3] and Chatterjee et al. [9], additional data that influences the human password-choice [33] might be helpful in determining the real vault. In fact, research has shown that background information about the user helps to guess passwords [6, 24] and to answer personal knowledge questions [26, 28]. We evaluated this assumption by using the available usernames or email addresses from PBVault to measure the effect on the ranking success. We considered an overlap between the username and password as an indicator for the vault being real, but preliminary experiments have shown that this is a weak indicator only. Therefore, we give this factor a small weight compared to the KL divergence, thus it basically resolves ties between two vaults with the same KL divergence. We converted both, the username and the password to lowercase and reverted some leetspeak transformations. If the (transformed) username was a substring of the password, we gave a score of 2; if the edit distance was below a threshold, we gave a score of 1.

Results from this experiment are summarized in Table 3. They show that the median rank for the real vault in NoCrack is 2.10 %, thus worsens the KL divergence attack result (median of 1.97 %). Note, for easier comparison we also report numbers for the static Markov NLE, which is introduced in Section 6. For Markov, we see a decrease of the median ranking result to 7.22 %, compared to the KL divergence attack with a median of 14.24 %.

5.2 Password Reuse

It is well known that users tend to reuse passwords across services. Reported numbers differ and range from around 10 % to around 60 % [12, 2, 10]. Hence, we expect to find this amount of reuse in the vaults as well. NoCrack simulates password reuse by decoding sub-grammars (see Section 2.3),

Table 3: Rank results based on four different attacks against entire vaults, where smaller numbers mean a more efficient attack. Note, for easier comparison we also report numbers for the static Markov NLE, which is introduced in Section 6. Decoy vaults are chosen from the NoCrack or Markov distribution, real vaults are chosen from the PBVault distribution. We list the results for varying classes of vault sizes.

Correlation Attack

PBVault	NoCrack: 30×10^6			Static Markov		
Vault Size	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
2-3	9.56 %	0.88 %	2.10 %	30.46 %	0.39 %	14.66 %
4-8	6.20 %	1.00 %	2.75 %	25.65 %	0.33 %	8.69 %
9-50	3.41 %	0.53 %	1.88 %	19.31 %	0.12 %	2.10 %
All	**6.36 %**	**0.92 %**	**2.10 %**	**25.08 %**	**0.18 %**	**7.22 %**

Reuse Attack

PBVault	NoCrack: 30×10^6			Static Markov		
Vault Size	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
2-3	9.59 %	0.95 %	2.11 %	31.48 %	0.49 %	16.82 %
4-8	5.97 %	0.96 %	1.94 %	26.86 %	0.17 %	9.72 %
9-50	3.14 %	1.12 %	1.74 %	24.83 %	1.18 %	12.69 %
All	**6.21 %**	**0.99 %**	**1.99 %**	**27.76 %**	**0.39 %**	**14.28 %**

Policy Attack

PBVault	NoCrack: 30×10^6			Static Markov		
Vault Size	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
2-3	3.43 %	0.70 %	1.42 %	20.03 %	0.48 %	15.29 %
4-8	2.44 %	0.72 %	1.34 %	17.47 %	0.18 %	8.96 %
9-50	1.74 %	0.85 %	1.26 %	17.07 %	1.20 %	11.24 %
All	**2.54 %**	**0.80 %**	**1.37 %**	**18.31 %**	**0.38 %**	**12.82 %**

Best Attack

PBVault	NoCrack: 30×10^6			Static Markov		
Vault Size	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
2-3	3.36 %	0.67 %	1.44 %	19.11 %	0.10 %	13.64 %
4-8	2.43 %	0.73 %	1.34 %	16.51 %	0.11 %	7.88 %
9-50	1.53 %	0.10 %	1.01 %	12.69 %	0.15 %	1.53 %
All	**2.43 %**	**0.56 %**	**1.31 %**	**16.15 %**	**0.17 %**	**6.54 %**

and for our Markov model-based NLE we implemented a similar solution. If implemented incorrectly, i. e., if the NLE outputs vaults with an unrealistic password reuse, this can be used as an indicator for real vaults as well. Similar to the correlation feature of Section 5.1, preliminary tests have shown that reuse is a relatively weak indicator, thus again, we use it with a small weight only, mostly breaking ties in the KL divergence.

For each vault, we calculate the reuse rate, i. e., given two randomly chosen passwords from the vault, what is the probability that these two are equal. In addition, we calculated a reuse rate for "similar" passwords, where similarity is measured by the Levenshtein edit distance for thresholds ranging from 1 to 5. This measure has been used before [2] in the context of reuse. Finally, we use a weighted average of these six reuse rates as the final indicator.

Results from these experiments are summarized in Table 3. We see that the results do not vary greatly. The median rank for the real vault with NoCrack is 1.99 %, thus does not improve the KL divergence attack result (median of 1.97 %). For Markov, we see the same, with a median ranking result of 14.28 % compared to the KL divergence attack with a median of 14.24 %. These results show that both, the NoCrack NLE as well as the Markov NLE, accurately simulate the available data in PBVault. However, there is no other data available to cross-check these results, and we expect this attack to perform better on fresh data, which may have a different behavior in terms of password reuse.

5.3 Password Policies

Many websites enforce password-composition policies on the passwords of its users. These rules differ from site to site and may change over time. Thus, it is difficult for an NLE to create passwords for a specific site that adhere to the imposed rules, without "overdoing" it and choosing unrealistically strong passwords. Bojinov et al. [3] surveyed policies for the Kamouflage system and found that the majority of large sites apply a minimum length criterion, e. g., at least 8 chars that should be considered by an NLE. We found that Chatterjee et al. [9] reported policy compliance for their UNIF NLE, which builds computer-generated passwords, but not for the main NLE, the vault-generating SG. In the following experiments, we assume that the user has at least one account stored in the vault that requires a minimum password length of 8. If decrypting the vault yields a shorter password for this specific account, we discard this vault as being non-compliant.

Some results from this experiment are summarized in Table 3. They show that the median rank for the real vault in NoCrack is 1.37 %, thus improving the KL divergence attack result (median of 1.97 %). For Markov, we see the same, with a median ranking result of 12.82 % compared to the KL divergence attack with a median of 14.24 %.

In principle, it is possible to prevent attacks based on the violation of password policies. One would need to keep track of password policies for the sites of interest, and modify the encoder to only generate compliant passwords. This task is, however, complicated by the fact that policies change over time, so we might need to re-encode the vault when a policy has changed. Also, password policies are not available in a machine-readable format, yet. However, a first solution for this problem has been proposed by Horsch et al. [16].

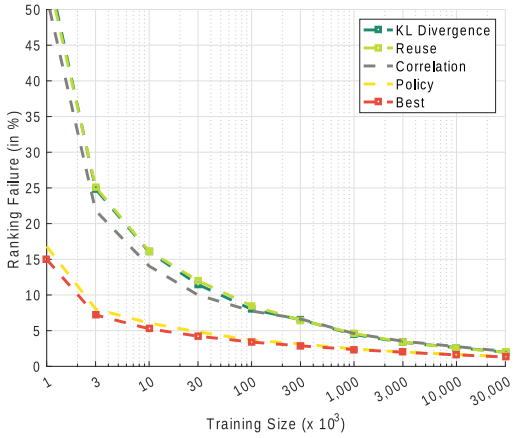

Figure 3: The median rank of all PBVault vault sizes (2-50).

5.4 Best: Combining the Factors

Finally, we combine the features Policy, Correlation, and KL divergence to an overall classifier. The results of this experiment are summarized in Table 3. They show that the median rank for the real vault in NoCrack is 1.31 %, thus improve the KL divergence attack result (median of 1.97 %). For Markov, we see the same, with a median ranking result of 6.54 % compared to the KL divergence attack with a median of 14.24 %. In Figure 3 we depict the summarized attack results against the NoCrack NLE for different training sizes, showing analog to Section 4.3 how an increased training set improves the ranking across all classifiers.

5.5 Further Remarks

Besides the already discussed structural differences of passwords, there are some more criteria that might be considered. First of all, knowing a leaked password from a website might be a great way to distinguish real from decoy vaults. Furthermore, as already mentioned by Chatterjee et al. [9] if a vault is stolen twice, the security falls back to that of a conventional PBE. The lack of real-world sample data does not allow experiments on the correlation of master passwords and corresponding domain passwords. Furthermore, the security pledge of NoCrack is somehow counterintuitive, as using a more unique (secure) self-chosen domain password facilitates distinguishing. Finally, if a website reports that an entered password was correct in the past like *Facebook* does, it might be a further help for an attacker to find the real vault, even though the user changed some domain passwords after the vault has been stolen.

6. ADAPTIVE NLES BASED ON MARKOV MODELS

Next, we describe a (static) NLE based on Markov models which has better properties than the PCFG-based NLE from NoCrack. Then we show how to turn this NLE into an adaptive NLE, and how this can improve the resistance against ranking attacks.

6.1 Static NLEs Based on Markov Models

Markov models are tools to model stochastic processes, widely used for natural language processing like automatic speech recognition. Recently, they have established themselves as an important tool for password guessing [23, 11, 30, 21] as well as measuring password strength [7]. In fact, an NLE based on Markov models was briefly tested by the authors of NoCrack [9] but dismissed in favor of a PCFG-based NLE.

Markov Models.

In an n-gram Markov model one models the probability of the next token in a string based on a prefix of length $n-1$. Hence, for a given sequence of tokens $c_1, \ldots, c_m$, an n-gram Markov model estimates its probability as

$$P(c_1, \ldots, c_m) \tag{1}$$
$$= P(c_1, \ldots, c_{n-1}) \cdot \prod_{i=n}^{m} P(c_i | c_{i-n+1}, \ldots, c_{i-1}).$$

We use 4-grams, which offer a good trade-off between memory consumption, speed, and accuracy [11, 21], and we use the full set of printable ASCII characters (95 characters). The required *initial probabilities* (IP) $P(c_1, \ldots, c_3)$ and *transition probabilities* (TP) $P(c_4 | c_1, \ldots, c_3)$ are estimated as the relative frequencies from training data, where we use the RockYou dataset. We apply simple additive smoothing to account for unseen n-grams. We train individual Markov models for each length in the range of 4 to 12 characters.

Encoding of a Password pwd.

The encoding is a (probabilistic) mapping from the set of passwords to bitstrings. To compute this encoding we fix an ordering of n-grams (e. g., the alphabetic ordering). For each transition probability, i. e., for each prefix of length 3, the fixed order gives us a partition of the interval $[0, 1)$ into segments whose lengths just correspond to the transition probabilities. For a given transition in *pwd*, we determine the corresponding segment $[a, b)$ (where $b - a = P(x_4 | x_1, \ldots, x_3)$). From this segment, we sample a uniformly chosen value s. This process is repeated for all transitions in the string *pwd*, and likewise for the initial probability and for the length of the password. Finally, this process yields a vector $\vec{S} = (s_1, \ldots, s_{\text{len}(pwd)-1})$ of $\text{len}(pwd) - 1$ values. The vector $\vec{S}$ can be encoded into a binary string using techniques similar to previous work [9].

Decoding of a Vector $\vec{S}$.

The (deterministic) decoding of a vector $\vec{S}$ is straightforward. The first value s_1 determines a length l for the password, after deciding in which segment it falls. In addition, this tells us which Markov model to use. The value s_2 determines the first 3-gram of *pwd*, and the subsequent values $s_3, \ldots, s_{l-1}$ determine the remaining transition n-grams.

Handling Vaults.

To simulate password reuse in a way similar to NoCrack's SG model, we generate vaults with related passwords. We determine the desired level of password reuse from the vaults in the PBVault set. We measured both exact reuse as well as reuse of similar passwords with a small Levenshtein distance (cf. [2]). The measured reuse rates are (48.52, 9.81, 4.17, 2.74, 2.08, 2.72) for Levenshtein distances of 0 to 5, respectively. Constructing vaults to match a given vector of reuse is not trivial, as there is a high level of interaction between the similar passwords. We construct vaults by selecting a "base password" and using that for a fraction of M_0 of passwords in the vault (exact reuse). Furthermore, we add a fraction of $M_1, \ldots, M_5$ of passwords with a Levenshtein distance of $1, \ldots, 5$ to the base password, respectively. The remainder of the passwords is filled up with unrelated passwords. All these passwords relate to each other, so the actual fraction of passwords with an edit distance of 1 will in general higher than M_0. We empirically determined values $\vec{M} = (M_i)$ such that the reuse rates match the empirical results given above. We used values $\vec{M} = (0.66, 0.06, 0.02, 0.01, 0.015)$, adding Gaussian noise with σ^2 of $(0.06, 0.034, 0.008, 0.004, 0.012)$, respectively.

The related passwords are determined by modifying the last transition probability, which models most of the modified reuse found in practice [10]. More sophisticated approaches can be tested for real-world implementations. For example, by considering more than the last n-gram position and more precisely simulating user behavior [32, 33].

Table 4: Rank results based on a KL divergence attack of entire vaults, where smaller numbers mean a more efficient attack. Decoy vaults are chosen from the *static* Markov or *adaptive* Markov distribution; real vaults are chosen from the PBVault distribution. For better comparability to previous work [9] we list results for varying classes of vault sizes.

KL Divergence Attack						
PBVault	Static Markov			Adaptive Markov		
Vault Size	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
2-3	31.50 %	0.45 %	16.83 %	42.21 %	12.66 %	33.01 %
4-8	26.88 %	0.17 %	9.71 %	39.55 %	11.36 %	32.32 %
9-50	24.81 %	1.21 %	12.49 %	38.63 %	4.46 %	36.83 %
All	**27.77 %**	**0.39 %**	**14.24 %**	**40.12 %**	**9.12 %**	**35.14 %**

6.2 Baseline Performance

First, we determine how well this static NLE performs. Therefore, we first rerun the experiments based on KL divergence that were presented in Sections 4.3 and 4.4.

6.2.1 Kullback–Leibler Divergence Attack

We first describe the results for *entire vaults*. The setup is similar to the setup described in Section 4.3, i. e., we choose real vaults from the PBVault list, and decoy vaults according to the distribution generated by the Markov model.

For determining the reference distribution, we slightly deviate from the previous approach of sampling the distribution empirically. There are two reasons for this. First, for Markov models, it is easy to extract an explicit description of the probability distribution from the code, namely by copying the IP and TP tables. This information is more accurate than an approximation based on sampling, and thus preferable. Second, it turned out that the probability distribution generated by Markov models is much more "spread out" than the distribution generated by NoCrack, which is more concentrated on fewer values. (For illustration we sampled $1.5\,M$ passwords for both distributions. We obtained $250\,k$ unique passwords for NoCrack and $1.25\,M$ unique passwords for Markov.)

The results are shown in Table 4. Comparable results for NoCrack can be found in Table 1. We see that the Markov NLE is substantially more robust against this attack, with an average rank of 27.8 % (NoCrack: 6.2 %) and median 14.2 % (NoCrack: 2.0 %). Interestingly, for the weak vaults, there is no big difference: $Q_{0.25}$ is 0.4 and 1.0, respectively.

The results for artificial vaults (independently chosen, size 10) can be found in Table 2. Here we see that Markov performs similar to NoCrack, with a median of 0.1 %, and only slightly better for the mean. The only exception is in the comparison with RockYou, for which it performs relatively close to random. For the other lists, the median is 0.1 % each, equal to NoCrack, only the median being slightly better with values between 5.3 % and 11.9 %.

6.2.2 Machine Learning Attacks

We also re-created the original attack based on machine learning, in order to check how well the Markov NLE fares against it. The NoCrack paper gives limited details only. In the *full* version [9] of the paper, they report their best performing ML engine was a Support Vector Machine (SVM) with a radial basis function kernel. They constructed four SVM-based classifiers, one for each of the following feature vectors: Repeat count (including numbers for the uniqueness of passwords, leetspeak transformations, capitalization, and tokens within a vault); Edit distance (including numbers for password pairs of different edit distances); n-gram structure (comprises percentage frequencies for the most popular n-grams, to characterize token reuse); Combined (combination of the three features above). We re-created the classifier with the information available and some deliberate tuning of the parameters, obtaining a classifier that shows similar performance to the original classifier. The results both for vaults and artificial vaults (called MPW in NoCrack's terminology) are shown in Table 5. To facilitate the comparison between the original SVM and our re-implementation, we list the results for NoCrack by Chatterjee et al. [9] as well. We see that both NoCrack and Markov perform very similar against this classifier and that the values are very similar to the reported ones for vaults and similar for MPW. However, this is mostly a sign for the ineptness of the used features, as we have seen that KL divergence is substantially better in ranking than this classifier.

Table 5: Ranking results for the re-created ML classifier, for NoCrack and static Markov, both with MPW and SG. To facilitate the comparison between the original SVM and our re-implementation, we list the results for NoCrack by Chatterjee et al. [9] as well.

Kind	ML Single (MPW)			ML Vaults (SG)		
Feature	[9]	NoCrack	S. Markov	[9]	NoCrack	S. Markov
Repeat ct.	0.60 %	4.71 %	3.43 %	37.40 %	38.45 %	45.30 %
Edit dist.	1.20 %	1.71 %	1.28 %	41.40 %	35.52 %	35.10 %
n-gram	0.60 %	2.83 %	2.13 %	38.50 %	38.90 %	32.72 %
Combined	1.00 %	0.54 %	0.40 %	39.70 %	37.80 %	40.91 %

6.3 Adaptive Construction

The basic idea of the adaptive construction is to modify the n-gram model such that it does not assign very low probabilities to passwords that actually appear in the vault. Otherwise, the appearance of a very improbable password in a candidate vault would be a strong signal for the real vault. Therefore, we modify the transition probabilities of an n-gram model as follows: (i) For each password occurring in the vault, we choose one n-gram from that password at random and increase the probability by multiplying it with 5. (ii) For all remaining n-grams, we increase probabilities by a factor of 5 with a probability of 20 %. (iii) Finally, we re-normalize all probabilities. Here, the constants 5 and 20 % are determined empirically to work well and provide reasonable security guarantees (see below). In the next subsection, we establish that the resulting (adaptive) NLE prevents online-verification attacks much better than previously seen (static) NLEs, and we discuss the security implications of the adaptive property in the subsequent section.

6.4 Performance of the Adaptive NLE

To evaluate the performance of the adaptive NLE, we rerun the same experiment as before using KL divergence and the PBVault dataset. The results are summarized in Table 4. They show that the real vault cv_{real} is ranked on average among the 40.12 % of the most likely vaults, thus increasing the amount of online guessing substantially. Note specifically that the 1st Quartile dropped from 0.39 % for the static Markov NLE to 9.12 % for the adaptive Markov NLE.

We tested several other boosting constants $(2, 4, 5, 6, 8, 10)$ which resulted in the following mean values (33.71 %, 39.38 %, 40.12 %, 40.36 %, 41.56 %, 43.4 %). We considered 5 to be suitable as beyond the improvements are small.

6.5 Security of the Adaptive NLE

We have to assume that an attacker is able to determine which n-grams have been boosted in the process, either because the attacker knows which corpus the original n-gram model has been trained on, or because the attacker is able to notice deviations from a "normal" distribution. In this case, it might be possible to infer information about the passwords stored in the vault. (In fact, if we only boosted n-grams for passwords in the vault and omitted the random boosting of other n-gram probabilities, then this would give rise to an easy and very efficient attack.) Next, we will show that the information that an attacker can infer is very limited.

Let B be the set of those n-grams that have been boosted (and this set is known to the attacker). Consider a password pwd, which might or might not be in the vault, and let N be the set of n-grams that pwd contains. Depending on if it is in the vault or not, the size of the intersection $N \cap B$ will change (it will be larger on average if pwd is in the vault, as in this case $N \cap B$ is guaranteed to be greater than 1). We consider the influence that learning the value $i := N \cap B$ has on the probability of seeing a password pwd_0. The ratio between $P(pwd = pwd_0)$ and $P(pwd = pwd_0 | i = i_0)$ can be estimated as follows, using Bayes' rule, writing $f(k; n, p)$ for the probability mass function of the Binomial distribution, writing $len(pwd)$ for the length of a password and $p = 0.2$:

$$\frac{P(pwd = pwd_0 | i = i_0)}{P(pwd = pwd_0)} = \frac{P(i = i_0 | pwd = pwd_0)}{P(i = i_0)}$$
$$= \frac{f(i_0 - 1; len(pwd) - 3, p)}{f(i_0; len(pwd) - 2, p)} < \frac{1}{1 - p} = 1.25$$

In other words, even an adversary knowing the exact set of boosted n-grams will increase the estimate of the probability of the correct password by a factor of 1.25, which has a very limited effect on the guessing behavior for an online guessing attack. The exact influence on password guessing depends on the precise distribution of passwords, or more specifically on the attacker's belief about the password distribution, and is thus hard to precisely quantify.

6.6 Limitations of the Adaptive NLE

Finally, we discuss some limitations of adaptive NLEs and the Markov-based adaptive NLE in particular. Adaptive NLEs demonstrate an interesting direction to overcome fundamental limitations of static NLEs, as we have demonstrated in Section 4. However, more work is required to better understand the mechanisms for providing adaptive NLEs and to quantify their security guarantees.

Our method for implementing adaptive NLEs based on n-gram models shows a first step towards realizing adaptive NLEs. The technique is straightforward, but better methods may exist. Note that we are unaware of an easy and promising way to base adaptive NLEs on PCFGs. The parameters that we used were determined empirically and seem to work well, but a more systematic treatment may reveal parameters with better overall performance. Altogether, we consider adaptive NLEs as still being work-in-progress.

So far we have considered vaults that do not change over time. If a new password is added to a vault, one possible way is to re-encode the entire vault as described in Section 6.3. Then the construction is vulnerable to the same intersection attack as NoCrack: Given the same vault before and after adding a new password, the correct master password will decrypt both vaults to the same set of passwords, whereas a wrong master password will decrypt to different decoys with high probability. It is unclear how this problem can be avoided, both for static and adaptive vaults.

7. CONCLUSION

There are various attacks against cracking-resistant vaults, of which distribution-based attacks are only one possible class. We showed that the proposed NoCrack NLE, which is based on a PCFG model, is too simple. We highlighted that the inability to distinguish, of the applied SVM-based machine learning engine, does not serve as a lower-bound security guarantee. Rather, we provided a distribution-based attack, utilizing KL divergence, which can distinguish real from decoy vaults. Additionally, we described further issues that need to be considered for the construction of a well-performing NLE. Next, we demonstrated that our proposed n-gram model outperforms the PCFG-based solution. Moreover, we introduced the notion of *adaptive* NLEs, where the generated distribution of decoy vaults depends on the actual passwords stored in the vault. This makes it unnecessary to "predict" the changes in password distributions over time, an inherent flaw of *static* NLEs. Unfortunately, the lack of real-world statistics and sample data on vaults does not exactly ease the situation in vault security research.

8. ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) Research Training Group GRK 1817/1.

9. REFERENCES

[1] AgileBits, Inc. 1Password Support: Technical Document – OPVault Format, Dec. 2012. https://support.1password.com/opvault-design, as of August 16, 2016.

[2] D. V. Bailey, M. Dürmuth, and C. Paar. Statistics on Password Re-use and Adaptive Strength for Financial Accounts. In *Conference on Security and Cryptography for Networks*, pages 218–235. Springer, 2014.

[3] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant Password Management. In *European Conference on Research in Computer Security*, pages 286–302. Springer, 2010.

[4] J. Bonneau. *Guessing Human-chosen Secrets*. PhD thesis, University of Cambridge, 2012.

[5] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Security and Privacy*, pages 553–567. IEEE, 2012.

[6] C. Castelluccia, A. Chaabane, M. Dürmuth, and D. Perito. When Privacy Meets Security: Leveraging Personal Information for Password Cracking. *CoRR*, abs/1304.6584, 2013.

[7] C. Castelluccia, M. Dürmuth, and D. Perito. Adaptive Password-Strength Meters from Markov Models. In

Network and Distributed System Security. Internet Society, 2014.

[8] R. Chatterjee. NoCrack Password Vault, Sept. 2015. https://github.com/rchatterjee/nocrack, as of August 16, 2016.

[9] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Cracking-Resistant Password Vaults using Natural Language Encoders. In *IEEE Security and Privacy*, pages 481–498. IEEE, 2015. Full Version: https://eprint.iacr.org/2015/788, as of August 16, 2016.

[10] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The Tangled Web of Password Reuse. In *Network and Distributed System Security*. Internet Society, 2014.

[11] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane. OMEN: Faster Password Guessing Using an Ordered Markov Enumerator. In *International Symposium on Engineering Secure Software and Systems*, pages 119–132. Springer, 2015.

[12] D. Florêncio, C. Herley, and P. C. van Oorschot. Password Portfolios and the Finite-Effort User: Sustainably Managing Large Numbers of Accounts. In *USENIX Security Symposium*, pages 465–479. USENIX Association, 2014.

[13] P. Gasti and K. B. Rasmussen. On the Security of Password Manager Database Formats. In *European Symposium on Research in Computer Security*, pages 770–787. Springer, 2012.

[14] J. Goldberg. On Hashcat and Strong Master Passwords as Your Best Protection, Apr. 2013. https://blog.agilebits.com/2013/04/16/1password-hashcat-strong-master-passwords/, as of August 16, 2016.

[15] A. Greenberg. Password Manager LastPass Got Breached Hard, June 2015. https://www.wired.com/2015/06/hack-brief-password-manager-lastpass-got-breached-hard/, as of August 16, 2016.

[16] M. Horsch, M. Schlipf, J. Braun, and J. Buchmann. Password Requirements Markup Language. In *Australasian Conference on Information Security and Privacy*, pages 426–439. Springer, 2016.

[17] A. Juels and T. Ristenpart. Honey Encryption: Security Beyond the Brute-Force Bound. In *Advances in Cryptology - EUROCRYPT*, pages 293–310. Springer, 2014.

[18] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, RFC Editor, Sept. 2000.

[19] A. Karole, N. Saxena, and N. Christin. A Comparative Usability Evaluation of Traditional Password Managers. In *International Conference on Information Security and Cryptology*, pages 233–251. Springer, 2010.

[20] Z. Li, W. He, D. Akhawe, and D. Song. The Emperor's New Password Manager: Security Analysis of Web-based Password Managers. In *USENIX Security Symposium*, pages 465–479. USENIX Association, 2014.

[21] J. Ma, W. Yang, M. Luo, and N. Li. A Study of Probabilistic Password Models. In *IEEE Security and Privacy*, pages 689–704. IEEE, 2014.

[22] D. McCarney, D. Barrera, J. Clark, S. Chiasson, and P. C. van Oorschot. Tapas: Design, Implementation, and Usability Evaluation of a Password Manager. In *Annual Computer Security Applications Conference*, pages 89–98. ACM Press, 2012.

[23] A. Narayanan and V. Shmatikov. Fast Dictionary Attacks on Passwords Using Time-space Tradeoff. In *ACM Computer and Communications Security*, pages 364–372. ACM, 2005.

[24] A. Peslyak. John the Ripper's Cracking Modes, the "Single Crack" Mode, May 2013. http://www.openwall.com/john/doc/MODES.shtml, as of August 16, 2016.

[25] S. Profis. The Guide to Password Security, Jan. 2016. http://www.cnet.com/how-to/the-guide-to-password-security-and-why-you-should-care/, as of August 16, 2016.

[26] A. Rabkin. Personal Knowledge Questions for Fallback Authentication: Security Questions in the Era of Facebook. In *USENIX Symposium on Usable Privacy and Security*, pages 13–23. USENIX Association, 2008.

[27] D. Reichl. KeePass Help Center: Protection against Dictionary Attacks, June 2016. http://keepass.info/help/base/security.html, as of August 16, 2016.

[28] S. Schechter, A. J. B. Brush, and S. Egelman. It's No Secret. Measuring the Security and Reliability of Authentication via "Secret" Questions. In *IEEE Security and Privacy*, pages 375–390. IEEE, 2009.

[29] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson. Password Managers: Attacks and Defenses. In *USENIX Security Symposium*, pages 449–464. USENIX Association, 2014.

[30] J. Steube. Introducing PRINCE. In *International Conference on Passwords*, pages 1–42. Springer, 2014.

[31] B. Stock and M. Johns. Protecting Users Against XSS-based Password Manager Abuse. In *ACM Symposium on Information, Computer and Communications Security*, pages 183–194. ACM, 2014.

[32] B. Ur, F. Noma, J. Bees, S. M. Segreti, R. Shay, L. Bauer, N. Christin, and L. F. Cranor. "I Added '!' at the End to Make It Secure": Observing Password Creation in the Lab. In *USENIX Symposium on Usable Privacy and Security*, pages 123–140. USENIX Association, 2015.

[33] R. Veras, C. Collins, and J. Thorpe. On the Semantic Patterns of Passwords and their Security Impact. In *Network and Distributed System Security*. Internet Society, 2014.

[34] M. Weir, S. Aggarwal, B. D. Medeiros, and B. Glodek. Password Cracking Using Probabilistic Context-Free Grammars. In *IEEE Security and Privacy*, pages 391–405. IEEE, 2009.

[35] F. Wiemer and R. Zimmermann. High-Speed Implementation of bcrypt Password Search using Special-Purpose Hardware. In *International Conference on ReConFigurable Computing and FPGAs*, pages 1–6. IEEE, 2014.