USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Practically Secure Honey Password Vaults: New Design and New Evaluation against Online Guessing

Haibo Cheng, Fugeng Huang, and Jiahong Yang, *Peking University;* Wenting Li, *Beijing Institute of Graphic Communication;* Ping Wang, *Peking University*

https://www.usenix.org/conference/usenixsecurity25/presentation/cheng-haibo

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

ARTIFACT EVALUATED
usenix ASSOCIATION
AVAILABLE

Practically Secure Honey Password Vaults: New Design and New Evaluation against Online Guessing

Haibo Cheng[1], Fugeng Huang[1], Jiahong Yang[1], Wenting Li[2], Ping Wang[1]
[1]*Peking University* [2]*Beijing Institute of Graphic Communication*

Abstract

Password vaults are used to manage multiple account passwords, encrypted with a master password. However, ciphertext stored on synchronization servers is vulnerable to leakage and offline guessing attacks, potentially compromising all accounts. Honey password vaults address this by generating decoy vaults for incorrect master passwords, making offline guessing infeasible and requiring online verification.

Existing studies on honey vaults rely on a *small* dataset of *only 276* vaults for model training and security evaluation, limiting their conclusions. More importantly, existing evaluations focus solely on the distinguishability between real and decoy vaults, overlooking practical security: How many accounts could be cracked via online guessing?

In this paper, we construct a large dataset of millions of vaults by aggregating numerous leaked password datasets. With the dataset, we employ advanced machine learning techniques for both decoy generation and identification. We show that various text classification algorithms, especially pre-trained models, significantly outperform existing attacks with distinguishing accuracy of 95.79%–83.75%. Further, we introduce a Transformer model that generates more plausible decoy vaults, no attacks achieve accuracy more than 64.35%.

We further assess the practical security of honey vaults against online guessing. Our new model achieves the best performance, only 0.51 accounts is cracked on average with 1,000 online attempts. By applying two simple measures, we enhance the scheme to a practical level: 1) using honey accounts for leakage detection, and 2) avoiding the encryption of passwords for websites with unlimited login attempts. These improvements reduce the cracked number to 0.11. We also offer new insights, such as that even a poor model can achieve notable practical security by using our measures.

*Wenting Li and Ping Wang are corresponding authors. Haibo Cheng and Ping Wang are also with Key Laboratory of High Confidence Software Technologies (PKU), Ministry of Education, China.

1 Introduction

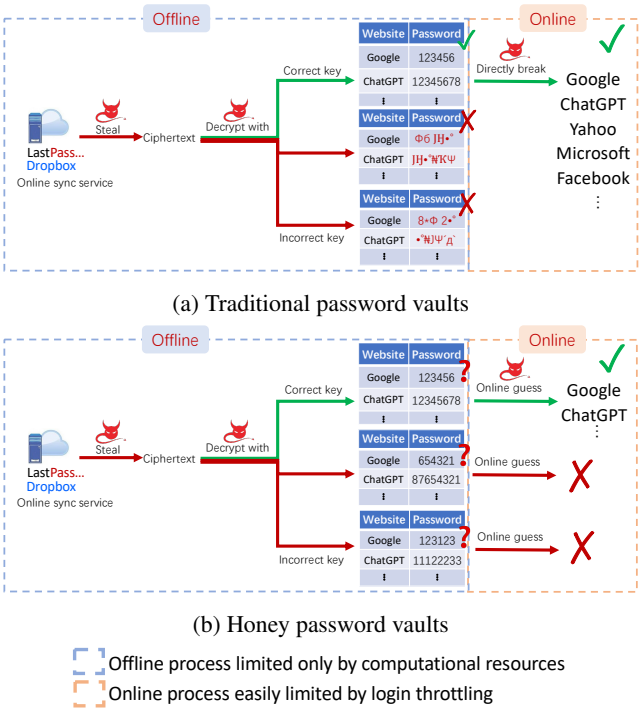
Password vaults are widely used by millions to store and manage multiple account passwords, significantly alleviating the burden on memory [29]. The user only needs to remember a master password, which encrypts the account passwords. The ciphertext is synchronized via a server for access across multiple devices (e.g., PCs and phones).

However, synchronization servers are vulnerable to leakage [13, 30], and the ciphertext is susceptible to offline guessing targeting the master password. As shown in Fig. 1a, if an attacker succeeds, all encrypted passwords are compromised. For instance, the 2022 LastPass breach [22] resulted in the loss of $5 million in cryptocurrency from LastPass users [35].

To mitigate this risk, honey password vaults, offering information-theoretic security, have been proposed[1]. Honey vaults generate plausible decoy vaults for incorrect master passwords, replacing traditional vaults that either fail or yield meaningless data. As shown in Fig. 1b, attackers cannot identify the real vault via *offline* guessing and decryption, but have to verify the vault's validity *online* through login attempts. With login throttling, attackers face limited guesses and find it difficult to compromise the vaults.

The concept of honey vaults was first introduced by Bojinov et al. [2], which involves N ciphertexts, including a real one and $N-1$ decoys. Later, honey encryption (HE) [19] was integrated into honey vaults [4] so that a single ciphertext produces decoy vaults for all incorrect keys. Several HE-based honey vault schemes [4, 6, 12] follow a similar design paradigm: 1) Estimate the probability distribution of messages (vaults) using suitable models; 2) Design a (randomized) encoder to transform messages sampled from the model into a uniformly random string (seed); 3) Combine the encoder with symmetric encryption so that the message is first encoded to

[1]Multi-factor authentication (MFA) is often employed [1, 21] for this threat. However, high-entropy factors (e.g., USB tokens) may be lost, preventing users from retrieving their passwords, while low-entropy factors (e.g., fingerprints) remain vulnerable to guessing attacks [5]. Besides, MFA can be integrated to enhance the security of honey vaults.



(a) Traditional password vaults

(b) Honey password vaults

Figure 1: The difference between traditional and honey password vaults in the view of attackers.

a seed and then encrypted into ciphertext. When an incorrect key is used, decryption results in a random seed, which, when decoded, produces a message sampled from the model.

The third step follows the general HE scheme by Juels and Ristenpart [19], which is *provably* secure if the encoder is secure. Cheng et al.'s approach [7] addresses the second step, transforming the probability model into an encoder with *provable* security if the probability model is indistinguishable from the real distribution. Further, Cheng et al. [6] enhanced the encoder's efficiency and introduced an incremental update mechanism for HE when updating the ciphertext, such as adding passwords to vaults. This mechanism with a secure encoder, is proven secure if both the encoder and the symmetric encryption are prefix-keeping. Consequently, the main challenge for honey vaults now lies in designing the probability model for password vaults.

The model plays a critical role in generating indistinguishable decoy vaults containing decoy passwords. For randomly generated passwords—which are recommended by password vaults for security—the model design is straightforward. However, for human-generated passwords, which users often prefer to store in vaults [23, 28], the modeling becomes significantly more challenging. Following prior work [4, 6, 7, 11, 12], we focus primarily on human-generated passwords.

Motivation. Prior to this work, *only a small* dataset of 276 vaults was used for model training and security evaluation [4, 6, 11, 12]. As a result, existing studies have relied on simple heuristic methods for both decoy generation and identification, potentially leading to dubious conclusions. It remains uncertain whether these honey vaults are secure in real-world applications with millions of users.

Moreover, existing security evaluations focus primarily on distinguishing real vaults from $N-1$ decoys [4, 6, 11, 12] (typically with $N = 1{,}000$) and use accuracy as the main metric. These evaluations do not assess the practical security of honey vaults against online guessing attacks. An open question remains: how many accounts could be compromised through online guessing?

1.1 Our Contribution

We create a large password vault dataset consists of 63,742,912 vaults by linking passwords from multiple leaked datasets to the same users. With this dataset, we enable to apply advanced text classification algorithms, including pre-trained models, to conduct distinguishing attacks. The pre-trained models achieve impressive accuracy, ranging from 95.79% to 83.75%, in distinguishing real and decoy vaults for existing honey vault schemes. This may be because models pre-trained on text can better capture the semantic features and patterns in passwords by leveraging their understanding of natural language. We also adapt a Transformer model, a cutting-edge and widely adopted approach in natural language processing, for vault distribution, showing clear advantages over existing models in defending against both our own and prior distinguishing attacks. No attack exceeds 64.35% accuracy, which is close to the 50% accuracy of trivial attacks.

More importantly, we evaluate the practical security of honey vaults against online guessing. Through formal analysis, we construct efficient online guessing strategies based on binary classifiers. Against such online guessing, our new vault scheme based on Transformer achieves the best. Within 1,000 online guesses, attackers only compromise 0.51 accounts on average. We further enhance our scheme with two simple measures. Measure I avoids encrypting passwords for websites with unlimited login attempts. Some websites lack proper login rate-limiting, giving attackers numerous chances to verify the correctness of the vault. Storing passwords for these sites in plaintext rather than ciphertext can limit the number of online attempts, significantly improving security. Measure II uses honey accounts for leakage detection. By incorporating honey accounts into the vault, we can detect ciphertext leakage through login attempts on these accounts and prevent further account compromise. Using the two measures greatly decrease the cracked account number to 0.11, making the practical security to a usable level.

We also provide some new insights: 1) Improving the probability model alone, without our measures, yields diminishing returns in practical security. Our scheme performs only slightly better than Cheng et al.'s scheme [6], despite significant advancements against distinguishing attacks in tradi-

tional evaluations. 2) Even a poor probability model can still achieve notable practical security by using our measures. For example, Cheng et al.'s scheme [12] remains effective against online guessing, even though it performs poorly in traditional evaluations. 3) Accuracy in traditional evaluations does not necessarily reflect practical security against online guessing.

2 Background and Related Work

As discussed in the Introduction, HE-based honey vault designs involve 1) probability models for vault distribution, 2) encoders for the models, and 3) HE schemes using the encoders. We review the research on these three components.

2.1 Honey Encryption

Honey Encryption (HE) was proposed by Juels and Ristenpart [19] to strengthen password-based encryption against password-guessing attacks. Unlike traditional symmetric encryption, HE is designed for a specific message distribution, p_{m} (e.g., for password vaults), and generates decoy messages following this distribution p_{m} when decrypted with incorrect keys. Juels and Ristenpart [19] introduced a general HE framework with a randomized encoder, known as the distribution-transforming encoder (DTE).

The core of the JR framework involves encoding the message sampled from p_{m} into a randomly uniform string (the seed), which is then encrypted. Decryption with incorrect keys yields random seeds, which, when decoded, produce messages sampled from p_{m}. This framework, with a secure encoder, has been proven to provide message recovery security, target-distribution semantic security in the random oracle model [16, 19], and target-distribution non-malleability in the ideal cipher model [16].

2.2 Encoder for Honey Encryption

The encoder for HE must ensure that decoding a random seed produces a message m following the distribution p_{m}, similar to sampling. Correspondingly, encoding m should produce a random seed, akin to inverse sampling. Thus, m corresponds to $p_{\mathrm{m}}(m)$ proportion of seeds, unlike traditional encoders (e.g., ASCII), where each character maps to a single code.

Based on inverse sampling, Juels and Ristenpart [19] introduced the IS-DTE, which is a general encoder scheme for arbitrary message distributions. Specifically, for a message distribution with cumulative distribution function F, IS-DTE assigns the seed interval $[F(m_{i-1}), F(m_i))$ on $\mathbb{R}$ for message m_i and discretizes it into the interval $[\mathrm{round}(|\mathcal{S}| \cdot F(m_{i-1})), \mathrm{round}(|\mathcal{S}| \cdot F(m_i)))$ on a discrete seed space $\mathcal{S}$. Here, round denotes the rounding function. To encode m_i, IS-DTE randomly selects a seed from this interval, following the inverse sampling method.

The security of the encoder is defined by the indistinguishability of the real message-seed pair (M_1, S_1) and the decoy pair (M_0, S_0): 1) M_1 is sampled from $p_{\mathrm{m}}(m)$, and S_1 is encoded from M_1; 2) S_0 is randomly selected from $\mathcal{S}$, and M_0 is decoded from S_0. IS-DTE is proven secure [19], with the only difference being the precision loss due to discretization.

In real-world applications, the message space $\mathcal{M}$ is typically exponential in size, and directly applying IS-DTE to the entire space results in exponential complexity. To address this, several HE applications, such as genomic data protection [15] and honey password vaults [4, 6, 12], use a divide-and-conquer algorithm to reduce the complexity. Cheng et al. [7] formalized and unified these encoder designs into a general framework called the probability-model-transforming encoder (PMTE). As the name suggests, PMTE is based on a probability model that estimates the message distribution. The core idea is to parse a message into a sequence of rules and encode each rule separately in a smaller space. For Markov models, PMTE encodes each character in the message, and for PCFG models, it encodes each production rule. Using IS-DTE for each rule results in the specific IS-PMTE design, which is proven secure if the probability model is indistinguishable from the message distribution [7].

It is important to note that some specific encoders [4, 12] used in honey vaults are not secure and can suffer from encoding attacks [7]. The main flaw is that the encoders ignore the ambiguity in the probability models. In certain models, a message can have multiple valid sequences of rules. For instance, in the Chatterjee-PCFG model [4], "password" can be generated by either $\mathtt{S} \to \mathtt{W}, \mathtt{W} \to$ password or $\mathtt{S} \to \mathtt{WW}, \mathtt{W} \to$ pass, $\mathtt{W} \to$ word. However, the flawed encoder chooses only one sequence, which makes the seeds corresponding to other sequences clearly decoys. More generally, the seed is selected from a subspace rather than the entire space, failing to guarantee seed uniformity.

PMTE addresses this flaw by randomly selecting the sequences. For probability models with high ambiguity, this random selection may lead to high complexity. As random selection is necessary to defend against encoding attacks, low-ambiguity models are recommended when possible [6]. Additionally, Cheng et al. [6] proposed an improvement to reduce the complexity for many probability models MD, following the pattern

$$\Pr_{\mathrm{MD}}((m_1, m_2, \ldots, m_n)) = \prod_{i=1}^{n} \Pr_{\mathrm{MD}}(m_i | (m_1, m_2, \ldots, m_{i-1})). \quad (1)$$

Their approach also uses a divide-and-conquer algorithm, where a partial sequence for m_i is randomly chosen for each i according to the conditional probability, rather than selecting the entire sequence for $(m_1, m_2, \ldots, m_n)$. Thus, IS-PMTE is highly efficient and applicable to many real-world scenarios, with provable security. In our paper, we assume that the encoders used in honey vault schemes are seed-uniform and do

not consider encoding attacks, as these schemes can employ IS-PMTE instead of the original encoders.

Additionally, Cheng et al. [6] proposed an incremental update mechanism/encoder for HE when updating the ciphertext, such as adding new passwords to vaults. Without this mechanism, previous schemes directly encrypt the new message using HE, creating correlations between the old and new messages under the correct key, while messages under incorrect keys remain independent. Such schemes are vulnerable to intersection attacks, which exploit these correlations when both the new and old ciphertexts are leaked. Cheng et al.'s mechanism encodes the modified part of the message into a seed s' and appends it to the previous seed s, forming a new seed $s||s'$. This new seed is then encrypted with the previous nonce. Using a prefix-keeping symmetric encryption scheme, the old ciphertext becomes a prefix of the new ciphertext. Thus, under incorrect keys, the old seed remains a prefix of the new ciphertext, creating similarity between the old and new messages. By this way, the mechanism prevents intersection attacks. Formally, the incremental mechanism with a secure encoder is proven secure if both the encoder and the symmetric encryption scheme are prefix-keeping.

Duan et al. [11] applied this incremental update mechanism to Chatterjee et al.'s [4] and Golla et al.'s [12] encoders, but found the mechanism vulnerable. However, the probability models used in these encoders do not follow the pattern in Eq. (1), meaning they are not prefix-keeping. This incompatibility renders the incremental update mechanism unsuitable for Chatterjee et al.'s and Golla et al.'s encoders. Duan et al. did not provide details on how the mechanism was applied to these encoders, which may explain the observed vulnerability.

2.3 Probability Models for Password Vaults

As discussed earlier, with the general HE scheme [19] and encoder scheme [6,7], the main remaining challenge for honey vaults is designing the probability model for password vaults.

Password vaults typically contain two types of passwords: human-generated and randomly generated. Although randomly generated passwords are secure and strongly recommended, users often prefer to retain some human-generated passwords in vaults to ensure access if the vault is unavailable or to facilitate manual entry on devices without vault support (e.g., Xbox) [23, 28]. While modeling randomly generated passwords is straightforward, modeling human-generated passwords remains a significant challenge. In this paper, we focus on human-generated passwords, following prior work on honey vaults [4,6,7,11,12], and unless otherwise specified, we use the term "passwords" to refer to them.

Studies on password guessing reveal key statistical features [9, 24–27, 33]: 1) passwords tend to be low-entropy, concentrated on a small set with meaningful patterns; 2) a user's multiple passwords often share similar characters or patterns. These patterns arise naturally from users' memory limitations. Based on these features, existing vault models typically consist of two components: 1) a single-password model that characterizes the distribution of individual passwords, and 2) a password-similarity/reuse model that captures similarities among a user's multiple passwords.

Chatterjee et al. [4] used a Probabilistic Context-Free Grammar (PCFG) for the single-password model and introduced sub-grammars for password similarity. Golla et al. [12] applied an n-order Markov model for the single-password model and proposed a reuse-rate model for password similarity. They also introduced an adaptive model that adjusted based on the plaintext. However, Cheng et al. [6] demonstrated that this adaptive model leaked plaintext information, thereby decreasing security instead of enhancing it.

Cheng et al. [6] followed the pattern in Eq. (1) to construct a vault model using multiple similar-password models. The multi-similar-password model was built as:

$$\begin{aligned} &\Pr_{\text{MSPM}}(pw_{i+1}|pw_1, pw_2, \ldots, pw_i) \qquad (2)\\ &= f(i)\Pr_{\text{SPM}}(pw_{i+1}) + \frac{1-f(i)}{i}\sum_{i'=1}^{i}\Pr_{\text{SSPM}}(pw_{i+1}|pw_{i'}), \end{aligned}$$

where $\Pr_{\text{SSPM}}$, $\Pr_{\text{SPM}}$, and $f(i)$ represent the single-similar-password model, the single-password model, and the unreused probability function, respectively.

Duan et al. [11] proposed two vault models to attack existing honey vault schemes. They claimed these models were derived from single-similar-password models in [25,33], following the pattern $\Pr(pw_2|pw_1)$. However, as shown in Eq. (2) and discussed in [6,25], constructing multi-similar-password models from single-similar-password models is not trivial. Duan et al. [11] provided no details on how their models were constructed. Furthermore, they did not use their new models to build new honey vault schemes. Therefore, we exclude Duan et al.'s vault models from our security evaluation of honey vault schemes.

Existing studies on honey vaults use a small dataset of only 276 vaults for model training, limiting the models to few parameters for password similarity. For example, Golla et al.'s reuse-rate approach relies on just five parameters trained from this dataset. These models heavily depend on single-password models, which have larger datasets for training. Consequently, they may fail to accurately capture the similarity among a user's passwords.

3 Our New Vault Dataset

Although commercial password vaults have experienced multiple breaches, no public password vault dataset is currently available. However, there are numerous publicly accessible password datasets available. We clean these datasets, identify passwords associated with the same users, and ultimately aggregate them into a large vault dataset.

3.1 Previous Vault Dataset

In honey vault studies, Pastebin is the only publicly available password vault dataset. However, it is not a real dataset leaked from password vault applications, but rather an aggregation of compromised passwords [4]. These passwords were collected via malware on various clients and subsequently leaked publicly. Chatterjee et al. [4] aggregated them into vaults based on similar or identical usernames. The dataset is relatively small, containing only 276 vaults, each with 2–50 passwords.

4iQ [25] is another aggregated password dataset that could be used as a password vault dataset. The original 4iQ dataset, discovered in a non-traditional web, contains 1.4 billion email-password pairs, likely compiled from various password breaches over time. Pal et al. [25] aggregated passwords belonging to individual users based on usernames or emails and used this dataset to study credential stuffing, where a user's password is predicted from previously leaked passwords. However, the original 4iQ excludes pairs where the same password and username appear across different websites. While this exclusion has minimal impact on password guessing research, it significantly affects studies on password vaults. Using 4iQ as a vault dataset means the vaults do not contain duplicate passwords, making them less representative of real-world vaults.

3.2 RaidForums Massive Database Collection

We primarily use the RaidForums Massive Database Collection (RMDC) to construct our password vault dataset. RMDC includes password leakage datasets from 182 websites, totaling 427 GB in size. This dataset is a compilation of password datasets distributed on RaidForums [34], a black-hat hacking forum. Various data breaches were shared on RaidForums, with many of them available for just a few dollars. After RaidForums was shut down, the dataset became publicly available via BitTorrent (BT), and we downloaded it.

3.3 Data Cleaning and Aggregation Method

RMDC consists of 182 individual files, each corresponding to a password breach from a specific website. From these, we manually selected 28 files that contained plaintext passwords, as shown in Table 6. These files are primarily in plain text but come in varying formats, including different types of user information and storage structures. Some files exhibit inconsistent formatting for different items, likely due to careless compilation by the original hacker, making the cleaning process both tedious and meticulous.

Data Cleaning. For each file, we manually identified the data format and developed parsing rules to extract the relevant information into a unified format (domain, username, email, password). In cases where a website used the email as the username, we left the username field empty. We excluded items that did not have an email address, as we rely on the email for aggregation.

Due to the complexity and errors in the original data formats, our parsing rules sometimes resulted in incorrect splitting of strings, which led to erroneous items. To address this, we exclude some unreasonable items: 1) Remove passwords with non-ASCII printable characters; 2) Ensure that the username length is at least 3 characters and the password length is between 4 and 25 characters; 3) Ensure the email contains an "@" symbol.

Data Aggregation. After cleaning the datasets into a unified format (domain, username, email, password), we identify accounts belonging to the same user by email and aggregate them into vaults. Initially, we attempted to aggregate accounts by username, but this led to incorrect vaults containing passwords from different users. For example, certain popular usernames, such as "bob", are associated with various passwords, which clearly do not originate from the same individual. To address this, we switched to aggregating by email. While this approach may result in the same user's passwords being split across multiple vaults, it significantly reduces the likelihood of mixing passwords from different users into the same vault. Since the latter error would have a more damaging impact on our evaluation, we opted to aggregate by email.

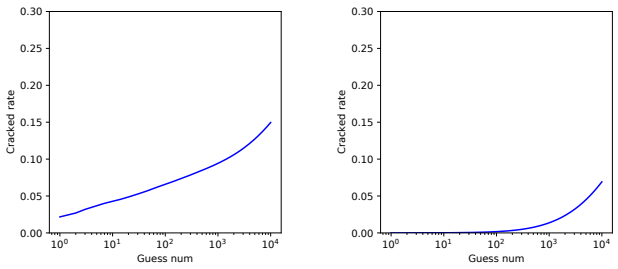
Even with email-based aggregation, a small number of vaults were unusually large, containing thousands of different passwords. This may be due to some websites allowing multiple accounts to share the same email address without verification, leading some users to use decoy emails. To address this, we excluded vaults within more than 1,000 accounts.

3.4 Master Passwords

The aggregated dataset merges account passwords used by the same user but lacks master passwords for the vaults. However, master passwords are critical for security evaluations in online guessing scenarios. Directly guessing the master password provides a trivial lower bound for cracking the vault. The lack of master passwords in previous evaluations has limited the focus to distinguishing between real and decoy vaults.

To address this, we simulate the master password distribution based on the distribution of account passwords. Intuitively, the master password is usually stronger than the account passwords, but there is still a lack of quantitative analysis on the strength of master passwords. Thus, we simulate two password distributions—normal and strong—to capture the lower and upper bounds of master password security.

For the simulation, we first combine passwords from various websites in the aggregated dataset to obtain the empirical password distribution. For Case I, with normal strength, we use the empirical distribution directly as the master password distribution. For Case II, with strong strength, we exclude the top 10,000 most probable passwords from the empirical distribution, normalize the remaining passwords, and use



(a) Normal strength (Case I) (b) Strong strength (Case II)

Figure 2: Guessing master passwords with normal and strong strength

Table 1: Vault composition statistics

Vault size	2-5	6-10	10-20	>20
Number of vaults	56834912	5687226	1082494	138280
Percent of total vaults	89.163%	8.922%	1.698%	0.217%
Avg. PW length	8.448	8.388	8.347	8.048
Avg. PWs per vault	2.531	7.459	13.245	49.924

this normalized distribution as the master password distribution. As shown in Fig. 2, the master passwords in Case II are considerably stronger than those in Case I. Later, we will demonstrate that using our new honey vault scheme, master passwords with normal strength can achieve nearly the same security as those with strong strength, indicating that users do not need to remember such strong master passwords.

Note that some user may leverage a master password related to the encrypted account passwords. This is not allowed by HE [19]. In such cases, the real message is related to the correct key, while a decoy message is independent of the corresponding incorrect key. This relationship allows an attacker to distinguish the real key. Achieving security in such scenarios remains an open question [6, 19]. Thus, we do not consider this scenario and follow prior work [4, 6, 7, 11, 12], which requires users to create a completely new master password.

3.5 Basic Statistics on Our Vault Dataset

Our constructed password vault dataset consists of 63,742,912 vaults. As shown in Table 1, for human-generated passwords: the average password length is approximately 8 characters; nearly 89% of vaults contain between 2 and 5 passwords, around 9% fall within the 6-10 range, 1.5% have between 11 and 20, and only 0.1% exceed 20 passwords. This distribution is consistent with the Pastebin vault dataset. As shown in Table 6, the dataset includes accounts from various websites, with the majority in Chinese or English, and a smaller proportion in Russian and Japanese. The websites cover various categories, including forums, games, and social platforms, reflecting a diverse range of real-world passwords.

Algorithm 1: The attack process with the stolen ciphertext of a password vault.

Input: a stolen ciphertext c, N master password guesses $\{mpw_i\}_{i=1}^{N}$, and a priority function p.

Offline:
  for $i \leftarrow 1$ **to** N **do**
    Decrypt c with mpw_i and get a vault $v_i = (pw_{i,1}, pw_{i,2}, \ldots, pw_{i,n})$.
  Sort $\{pw_{i,j}\}_{i,j}$ in descending order of $p(pw_{i,j})$ and get a list $L = \{(j, pw_{i,j})\}_{i,j}$. /* Different vaults may contain the same password, and these passwords will be merged into one during sorting. */
Online:
  for $(j, pw_{i,j}) \in L$ **do**
    Try logging into the j-th account with the password pw_i.
    If successful, exclude all the guesses $(j, pw_{i',j})$ from the list L for the j-th account. Otherwise, exclude all guesses $(j'', pw_{i',j''})$ in $v_{i'}$, including $(j, pw_{i,j'})$ where $pw_{i,j} = pw_{i,j'}$.

4 Our New Attacks

We first provide the security model, and formally analyze the optimal online guessing attack to compromise user accounts instead of just distinguishing real and decoy vaults. Further, we leverage some advanced text classification algorithms to construct distinguishing attacks and online guessing attacks.

4.1 Security Model

We focus on the threat posed by the leakage of ciphertexts in honey vault schemes. In accordance with Kerckhoffs's principle, we assume the attacker gains access to the program of the honey vault, including the encoder and symmetric encryption. This assumption is practical, as the program is publicly available to all users and typically stored alongside the ciphertext for user access. Note that the probability model (or encoder) in honey vault schemes requires a training set, often a public dataset containing many real vaults. We assume the attacker is aware of this training set and can leverage it to learn statistics from the real vaults within it. Therefore, both the attacker and defender share the same knowledge regarding the real vault distributions, including the same set of samples. Meanwhile, the attacker has full knowledge of the decoy distribution, facilitated by access to the encoder in the program. Additionally, the attacker is assumed to know the master password distribution, enabling optimal guessing attacks on master passwords.

With access to the ciphertext and the program, the attacker can decrypt the ciphertext using a dictionary of master passwords $\{mpw_i\}_{i=1}^{N}$, obtaining a vault v_i for each mpw_i, and may attempt online guessing to compromise the accounts. In previous studies [4, 6, 7, 11, 12], the goal of the attacker is to identify real vaults with the fewest online attempts. In contrast, we assume that the attacker seeks to compromise

the maximum number of accounts with the fewest online attempts, reflecting practical security in real-world scenarios.

We characterize this attack process using Algorithm 1, which incorporates a priority function p that indicates the priority $p(pw)$ of attempting to guess the password pw of an account. Unlike Algorithm 1 in [6], which sorts vaults, our Algorithm 1 sorts account passwords with finer granularity. More importantly, unlike the attackers in previous studies who merely sort, the attacker in our Algorithm 1 can adaptively adjust their guesses based on previous results. Specifically, if an account password is verified as correct or incorrect, all vaults with incorrect passwords for that account are treated as decoys. Thus, in contrast to previous studies, our Algorithm 1 models a more powerful and practical attacker scenario, reflecting real-world conditions more accurately.

It is important to note that we do not assume a practical attacker has complete knowledge of the real vault distribution. If such knowledge were available, it could directly be used to construct perfectly secure honey vaults that generate totally indistinguishable decoy vaults, thereby ending the study of design probability models for password vaults. Hence, we only assume a practical attacker has access to a dataset of real vaults, not the exact distribution of real vaults. However, in our theoretical analysis, we assume an optimal attacker has complete knowledge of the real vault distribution, and build efficient practical attackers based on this analysis. This follows traditional evaluation methods in HE applications.

4.2 Optimal Online Guessing in Theory

Intuitively, the optimal strategy for online guessing is to attempt to compromise the j-th account with the password guess $pw_{i,j}$ in decreasing order of the conditional probability of $pw_{i,j}$ given the ciphertext. This differs from the optimal strategy used in previous studies for distinguishing attacks with the conditional probability of the vault v_i [6, 11].

To formally analyze the conditional probability of a password guess given the ciphertext, we introduce some notations. For clarity and consistency, italic capital letters to denote random variables (e.g., X), italic lowercase letters (e.g., $x = 0$) represent specific values, and expressions like $X = x$ represent random events. We define the random variables for the (real) master password, vault, seed, and ciphertext as MPW, V, S, and C, respectively. The vault V consists of n passwords (PW_1, $PW_2, \ldots, PW_n$). Therefore, $MPW = mpw_i$ denotes the event that a user's real master password is mpw_i, i.e., the user selects mpw_i as the master password. Similarly, $V = v_i$, $S = s_i$, and $C = c$ denote the corresponding events. Let $v_i = (pw_{i,1}, pw_{i,2}, \ldots, pw_{i,n})$. For consistency with the subscript i, we define mpw_i, v_i, s_i, and c such that s_i is decrypted from c using mpw_i, and v_i is decoded from s_i. We further denote the real master password distribution $\Pr(MPW = mpw_i)$ as $\Pr_{\text{MPW}}(mpw_i)$, the real vault distribution $\Pr(V = v_i)$ as $\Pr_{\text{real}}(v_i)$, the decoy vault distribution as $\Pr_{\text{decoy}}(v_i)$ (i.e., the probability of getting v_i by decoding a random seed), respectively.

Theorem 1. *If the encoder is seed-uniform, then*

$$\Pr(PW_j = pw_{i,j} | C = c) = k \cdot \sum_{i':pw_{i',j}=pw_{i,j}} \Pr_{\text{MPW}}(mpw_{i'}) \frac{\Pr_{\text{real}}(v_{i'})}{\Pr_{\text{decoy}}(v_{i'})}, \quad (3)$$

where k is a constant independent of i.

Proof.

$$\begin{aligned}
&\Pr(PW_j = pw_{i,j} | C = c) \\
=&\sum_{i'} \Pr(V_{i'} = v_{i'}, MPW = mpw_{i'}, PW_j = pw_{i,j} | C = c) \\
=&\sum_{i':pw_{i',j}=pw_{i,j}} \Pr(MPW = mpw_{i'} | C = c) \\
=&k \cdot \sum_{i':pw_{i',j}=pw_{i,j}} \Pr_{\text{MPW}}(mpw_{i'}) \frac{\Pr_{\text{real}}(v_{i'})}{\Pr_{\text{decoy}}(v_{i'})}.
\end{aligned}$$

The last equation follows from Theorem 2 in [6]. □

In Eq. (3), k depends on the traditional encryption component used in HE. For example, for PBE with l_1-bit IVs and l_2-bit salts, $k = \frac{1}{2^{l_1+l_2}}$. Theorem 1 considers only seed-uniform encoders, as we primarily focus on honey vault schemes that use such encoders (see the discussion in Section 2.2). For general encoders, Theorem 1 in [6] provides a similar result by replacing $\frac{1}{\Pr_{\text{decoy}}(v_i)}$ in Eq. (3) with $\Pr_{\mathsf{encode}}(s_i \mid v_i)$, leading to a similar formulation of our theorem.

The sum in Eq. (3) is taken over all i' such that $pw_{i',j} = pw_{i,j}$, i.e., all vaults containing $pw_{i,j}$ as the j-th password. Unlike our approach, the analysis in [6] focuses solely on $\Pr(MPW_i = mpw_i | C = c)$, and the study in [11] examines only $\Pr(V_i = v_i | C = c)$. In [6], Cheng et al. stated that the probability of two vaults v_i and $v_{i'}$ being identical is very low, especially with large vault sizes, implying $\Pr(MPW_i = mpw_i | C = c) \approx \Pr(V_i = v_i | C = c)$. While this holds for vaults, it does not apply to passwords. Multiple vaults contain the same password for the same website. Thus, summing over all relevant vaults is essential.

Optimal Online Guessing. By Theorem 1, the optimal guessing strategy follows the decreasing order of the optimal priority function:

$$p_{\text{opt}}(pw_{i,j}) = \sum_{i':pw_{i',j}=pw_{i,j}} \Pr_{\text{MPW}}(mpw_{i'}) \frac{\Pr_{\text{real}}(v_{i'})}{\Pr_{\text{decoy}}(v_{i'})}.$$

4.3 Online Guessing Based on Binary Classifiers

Text classification algorithms have been extensively developed in natural language processing (NLP). It is natural to use binary classifiers for distinguishing attacks, which identify

the real vault among $N-1$ decoys. Specifically, a classifier D outputs a probability $\Pr_D(v_i)$ for each vault v_i, indicating the likelihood that v_i is real. The classification accuracy reflects the effectiveness of distinguishing attacks.

However, binary classifiers cannot be directly applied to online guessing as in Eq. (3). One might consider using the output probability $\Pr_D(v)$ as $\frac{\Pr_{\text{real}}(v)}{\Pr_{\text{decoy}}(v)}$, but this approach is inefficient due to the mismatch in mathematical structure. Fortunately, through simple analysis, we propose a method to construct online guessing by estimating $\frac{\Pr_{\text{real}}(v)}{\Pr_{\text{decoy}}(v)}$ with $\frac{\Pr_D(v)}{1-\Pr_D(v)}$.

Theoretical Analysis. While it may seem intuitive to use $\frac{\Pr_D(v)}{1-\Pr_D(v)}$ to estimate $\frac{\Pr_{\text{real}}(v)}{\Pr_{\text{decoy}}(v)}$, it is important to note that $\Pr_D(v)$ does not exactly equal $\Pr_{\text{real}}(v)$, and $1-\Pr_D(v)$ is not precisely $\Pr_{\text{decoy}}(v)$.

Consider a binary classification problem as a stochastic process with two distributions, p_0 and p_1. A random variable B is selected from $\{0,1\}$ according to a Bernoulli distribution p_{b}, and a sample X is drawn from the distribution p_B. Given a sample x, a classifier outputs the probability that x is from p_1. For an optimal classifier D_{o}, the output probability is:

$$\Pr_{D_{\text{o}}}(x) = \Pr(B=1|X=x) = \frac{p_1(x)p_{\text{b}}(1)}{p_1(x)p_{\text{b}}(1)+p_0(x)p_{\text{b}}(0)}.$$

Thus,

$$\frac{\Pr_{D_{\text{o}}}(x)}{1-\Pr_{D_{\text{o}}}(x)} = t\cdot\frac{p_1(x)}{p_0(x)},$$

where $t=\frac{p_{\text{b}}(1)}{p_{\text{b}}(0)}$ is independent of x. Since the real classifier D is an estimate of $\Pr_{D_{\text{o}}}$, $\frac{\Pr_D(x)}{1-\Pr_D(x)}$ serves as an estimate of $t\cdot\frac{p_1(x)}{p_0(x)}$, where the constant t can be disregarded in ranking.

Online Guessing Based on Binary Classifier. For a classifier D, online guessing follows the priority function:

$$p_D(pw_{i,j}) = \sum_{i':pw_{i',j}=pw_{i,j}} \Pr_{\text{MPW}}(mpw_{i'})\cdot\frac{\Pr_D(v_{i'})}{1-\Pr_D(v_{i'})}.$$

From Eq. (3), the constants t and k can be neglected as they do not affect the ranking of password guesses $pw_{i,j}$.

Trivial Online Guessing. For a trivial classifier D_{d}, which does not use any information from v and outputs a constant, the online guessing is governed by the trivial priority function:

$$p_{\text{triv}}(pw_{i,j}) = \sum_{i':pw_{i',j}=pw_{i,j}} \Pr_{\text{MPW}}(mpw_{i'}).$$

4.4 Text Classification Algorithms

We employ various text classification algorithms to implement our online guessing attacks, as shown in Fig. 3, and select some ones with good performance for demonstration.

For vaults containing multiple passwords, we concatenate the passwords using tab characters as delimiters, as passwords do not include tab characters. This allows us to treat the vault as text and apply text classification algorithms.

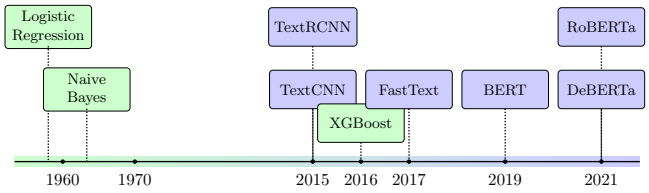


Figure 3: Classifiers we use

In our experiments, the deep learning models consistently outperformed traditional statistical learning methods such as Logistic Regression. As is typical in text classification using deep learning models, the input text is first tokenized into a sequence of tokens, where each token represents the smallest semantic unit, such as a character, word, or subword [10, 18], depending on the specific model requirements. This sequence is then processed by a deep neural network, followed by a fully connected layer with an output dimension of 2, which produces a classification probability distribution via a Softmax function.

The following four deep learning models demonstrate strong performance in online guessing, with experimental results presented in Section 6.

FastText [18] is a lightweight and efficient embedding-based method that incorporates subword information to capture semantic similarities between words. Compared to traditional n-gram-based models that treat words as discrete tokens, FastText models internal word structure and performs well on resource-constrained tasks.

TextCNN [20] applies convolutional filters of varying kernel sizes to extract local features from word embeddings, effectively capturing n-gram-like patterns. Its architecture enables the identification of salient phrases and local dependencies within a fixed context window.

DPCNN (Deep Pyramid Convolutional Neural Network) [17] enhances standard CNN models by introducing deep residual connections and hierarchical down-sampling. This design significantly increases the receptive field, enabling effective modeling of long-range dependencies and high-level semantic abstraction.

BERT [10, 32] represents a paradigm shift by leveraging the Transformer architecture and pretraining on large corpora to learn bidirectional contextual representations. For classification tasks, the [CLS] token's embedding is passed to a fully connected layer. Among the variants we evaluated, DeBERTa-v3 [14] achieved the best performance.

4.5 Existing Distinguishing Attacks

All existing attacks in the current study of honey vaults are distinguishing attacks, which rely solely on a priority function for vaults to differentiate one real vault from $N-1$ decoys.

Chatterjee et al. used Support Vector Machines (SVM) for distinguishing attacks [4]. Golla et al. employed Kullback-Leibler (KL) Divergence between a vault's empirical password distribution and the estimated real password distribution to prioritize vaults [12].

Cheng et al. [6] conducted a theoretical analysis to estimate $\Pr_{\text{real}}$ and $\Pr_{\text{decoy}}$ individually, using their ratio as the priority function to simulate the optimal distinguishing attack. They estimated $\Pr_{\text{real}}$ and $\Pr_{\text{decoy}}$ over small spaces, such as individual passwords and some password-similarity Bernoulli features, and then multiplied these estimators to produce an overall estimate for the entire vault space.

Duan et al. [11] used two password-reuse models, Pass2path [25] and TarGuess-II [33], to construct a vault model VM and estimate $\Pr_{\text{real}}$ using $\Pr_{\text{VM}}$. They then simulated the optimal distinguishing attack with $\Pr_{\text{decoy}}$. However, as discussed in Section 2.3, they did not provide sufficient details for reproducing their construction. They also attempted to incorporate the master password into distinguishing attacks. Traditional evaluations for distinguishing attacks only generate N vaults (one real and $N-1$ decoys), with the decoys created from random seeds, but without providing the corresponding master passwords to attackers. Duan et al. used $\sqrt[n]{\Pr_{\text{VM}}(v_i)}$ to estimate $\Pr_{\text{MPW}}(mpw_i)$, which is unreasonable, but unfortunately, there is no better solution within the traditional evaluation framework. In contrast, our new evaluation includes the master password information, which aligns better with practical security scenarios.

Online Guessing Based on These Distinguishing Attacks. Chatterjee et al.'s SVM can be adapted to online guessing using our method. Cheng et al. and Duan et al.'s attacks estimate the ratio $\frac{\Pr_{\text{real}}(v_i)}{\Pr_{\text{decoy}}(v_i)}$, which can be directly transformed into online guessing as shown in Eq. (3) with $\Pr_{\text{MPW}}(mpw_i)$. Golla et al.'s KL divergence attacks do not directly match Eq. (3). However, we can treat the KL divergence as an estimator for $\frac{\Pr_{\text{real}}(v_i)}{\Pr_{\text{decoy}}(v_i)}$, and apply the same approach as Cheng et al. and Duan et al. Thus, for Golla et al., Cheng et al., and Duan et al.'s distinguishing attacks, the priority functions p_{v} for vaults, can be converted to priority functions for passwords as:

$$p(pw_{i,j}) = \sum_{i':pw_{i',j}=pw_{i,j}} \Pr_{\text{MPW}}(mpw_{i'}) p_{\text{v}}(v_{i'}).$$

4.6 Previous Security Model and Metrics

In the previous security model for honey vaults [4,6,7,11,12], the attacker has the similar knowledge and source as that in our model but they aim to identify the the real vault from $N-1$ decoys.

Two security metrics are used: the rank cumulative distribution function (RCDF) proposed in [7] and distinguishing accuracy proposed in [4]. We find that these two metrics are equivalent to commonly used metrics in machine learning for binary classification: the Receiver Operating Characteristic (ROC) curve and the area under the ROC curve (AUC). In the following, we first present the metrics and then demonstrate their equivalence.

In traditional evaluation, the attacker ranks N vaults, consisting of one real vault and $N-1$ decoys, using the priority function p. The rank $\hat{R}(v)$ of the real vault v provides an indication of its security. To create a metric independent of N, the rank $\hat{R}(v)$ is typically represented in relative form, $\hat{r}(v) \in [0,1]$, as $\frac{\hat{R}(v)-1}{N-1}$ [6,7][2]. RCDF is the cumulative distribution function (RCDF) of relative ranks of real vaults. The average relative rank $\bar{r}$ is the complement of the accuracy α in distinguishing one real vault from one decoy [6,7], i.e., $\bar{r} = \alpha$.

Theorem 2. *The RCDF F for the priority function p is equivalent to the ROC curve F' with the weight function p.*

Proof. For simplicity, we omit the cases where two vaults have the same priority or weight. Handling these cases can be done using the same proof with some straightforward but tedious techniques.

Using t as the threshold for classification yields the point $(\text{FPR}(t), \text{TPR}(t)) = (\Pr(p(V_{\text{decoy}}) \geq t), \Pr(p(V_{\text{real}}) \geq t))$ on the ROC curve, where V_{decoy} and V_{real} are random variables sampled from the decoy and real vault distributions, respectively. By definition, $F(x) = \Pr(r(V_{\text{real}}) \leq x)$, and $r(v_{\text{real}}) = \Pr(p(V_{\text{decoy}}) \geq p(v_{\text{real}}))$. For a threshold $t = p(v)$, $\text{FPR}(t) = \Pr(p(V_{\text{decoy}}) \geq t) = \Pr(p(V_{\text{decoy}}) \geq p(v)) = r(v)$, and $\text{TPR}(t) = \Pr(p(V_{\text{real}}) \geq t) = \Pr(r(V_{\text{real}}) \leq r(v)) = F(r(v))$. Thus, F is equivalent to F'. □

Naturally, the accuracy is equivalent to AUC.

4.7 Other Security Models

This paper primarily considers attackers with access to the vault ciphertext who aim to compromise as many accounts as possible. However, an attacker may focus on a specific high-value target, such as a financial account. To maximize the success rate against the target, the attacker may first verify other accounts to eliminate incorrect password candidates, and then attempt to compromise the target account using the remaining candidates.

Furthermore, an attacker may obtain a leaked password from an unrelated data breach and use it to exclude decoy vaults that are inconsistent with the compromised password. To prevent such attacks from occurring, Cheng et al. [6] propose a method that monitors password and vault breaches and proactively updates leaked passwords and vaults.

5 Our New Scheme for Honey Vaults

We use a decoder-only Transformer model to construct a probability model for password vaults. This model is unambigu-

[2] In some studies [4, 12], $\frac{\hat{R}(v)}{N}$ is used, which yields a biased estimate of the relative rank. The expression $\frac{\hat{R}(v)-1}{N-1}$ provides an unbiased estimate.

ous, allowing us to naturally build an encoder for HE using the method by Cheng et al. [6, 7] and implement a honey vault scheme based on JR's general HE scheme [19]. We further leverage two simple measures to enhance the security of honey vaults to a usable level.

To avoid ambiguity, we clarify that the term "encoder/decoder" appears in both HE and Transformer with different meanings. In Section 5.1, "encoder/decoder" refers exclusively to the Transformer context, unless otherwise stated. In the rest of the paper, it refers to the HE context.

5.1 Our New Vault Model

We leverage the Transformer model to construct our vault model, as it represents a cutting-edge and widely adopted architecture in NLP, particularly well-suited for capturing long-range dependencies within sequences. Unlike prior vault models [4, 6, 12] that rely on simple methods for password similarity, the Transformer with its attention mechanism can effectively model both the distribution of individual passwords and the similarity between a user's passwords.

Our New Model. We use the standard decoder-only Transformer architecture, which is adopted by most state-of-the-art generative models [3]. The password vault is represented in a generative sequence format as

$$\texttt{<SEP>password1<SEP>password2<SEP>}\cdots,$$

where <SEP> is a special character serving as a delimiter between different passwords. As shown in Fig. 4, the model estimates the conditional probability of each character given the preceding characters. Therefore, it naturally forms a multi-similar-password model for

$$\Pr(pw_i|pw_1, pw_2, \ldots, pw_{i-1}),$$

and, ultimately, a vault model.

The vocabulary consists of 98 tokens, including 95 printable ASCII characters (codes 32 to 127) and a special characters: <SEP>. The transformer model has a hidden dimension of 768, with fully connected layers of size $d \times 4d$, following the standard architecture and has 85 M parameters. It includes 12 attention heads and 12 Transformer layers. The model is trained using gradient accumulation over 5 epochs, with a learning rate of 1×10^{-4}, 1000 warm-up steps, gradient clipping at 0.5, and a batch size of 32×2.

Note that the maximum password length is set to 25 during data cleaning. To ensure that the model generates consistent passwords, we directly truncate passwords that reach 25 characters by setting the probability of <SEP> to 1. In real-world applications, the maximum length can be adjusted as needed.

New Honey Vault Scheme. Our model naturally lends itself to constructing a honey vault scheme. Unlike Cheng et al.'s model [6], our model captures the conditional probability of characters in pw_i one by one, eliminating ambiguity. Thus, we can build an efficient encoder for HE using Cheng et al.'s methods [6, 7] without random path selection, and further integrate it into a honey vault scheme using JR's HE framework [19]. Since our model follows Eq. (1), our HE encoder is prefix-keep, and can leverage Cheng et al.'s incremental update mechanism to achieve provable update security.

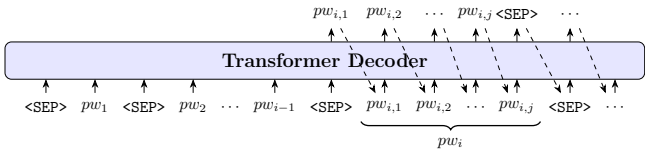


Figure 4: Decoder-only Transformer model

5.2 Two Simple Measures

Beyond improving the probability model for honey vault schemes, we leverage two simple but very efficient measures to enhance the security of honey vaults. The two measures are not limited to our scheme and can be applied to general honey vault schemes.

Measure I avoids using the master password to encrypt passwords for websites with unlimited login attempts. Although most popular websites implement login rate-limiting to mitigate online attacks, some still lack adequate protection. For accounts on such unlimited websites, where attackers can make numerous login attempts, storing passwords in a honey vault could pose a significant risk. For these accounts, we strongly recommend using randomly generated passwords and storing them in plaintext (which will later be encrypted by the sync server). This measure limits the number of online guessing attempts by an attacker who steals the vault, significantly improving security.

Randomly generated passwords ensure that attackers cannot use them to identify the real vaults, as there is no similarity to human-generated passwords. Additionally, this method limits the breach risk from these unlimited websites, as they are likely to have weak password storage protection.

Storing such passwords in plaintext may seem insecure, as attackers can directly compromise the accounts. One might consider using another key to encrypt these passwords: 1) Using a random key stored in the honey vault: This method allows attackers to verify the correctness of the key through unlimited online attempts, further aiding in the identification of the real vault. 2) Using another password: This increases the memory burden for users, negatively impacting usability. Moreover, attackers may gain an advantage due to the reuse relationship between the password and the real vault. 3) Using a part (e.g., 4 bits) of the master password: This allows attackers to exclude most decoys (93.75% for 4 bits) through unlimited online attempts, drastically reducing security.

Storing these passwords in plaintext minimizes the damage caused by unlimited online attempts. Accounts on such

websites typically have limited value, so attackers gain minimal benefit. Moreover, as the attacker obtains the plaintext passwords, they may compromise the accounts first before decrypting the ciphertext. Thus, login attempts on these accounts could serve as indicators of potential ciphertext leakage.

Measure II involves using honey accounts within the user's real vault to detect ciphertext leakage. The core idea is to register accounts on various websites and store them in the user's real vault. These accounts are not used by legitimate users, so any login attempts on these accounts can serve as indicators of potential ciphertext leakage. We use a server, referred to as the monitor, to track login attempts on these honey accounts and send leakage notifications to the user, prompting them to change their vault passwords to prevent further account compromises.

There are two key elements to this measure: 1) how to select the websites and monitor the login attempts; 2) how to generate the usernames and passwords.

To monitor leakage, the websites for honey accounts must provide login notifications. Some websites, like Google, notify the user via email, while others, such as GitHub, provide detailed login logs on their web pages. The monitor can use any form of login notice, not just email. However, in practice, not every website provides such notifications. This creates a potential risk, as the attacker may identify real accounts based on whether the website offers such notices. In this case, the attacker can first target the real accounts that do not trigger leakage alerts and then proceed to attack the remaining set, which contains a mixture of real and honey accounts.

To ensure the indistinguishability of real and honey accounts, the usernames and passwords for honey accounts should follow the same creation habits as the real accounts. For perfect security, the user can manually create them. For improved usability, the passwords can be generated automatically by sampling from a probability distribution, and the usernames can be reused from the user's other accounts.

The leakage alarms may occasionally be triggered falsely, either by the user or the attacker. The user may accidentally log in to one of these accounts, triggering a false alarm. To address this, we recommend notifying the user with a message such as: "If this was not you logging in, your password vault may have been leaked." An attacker, who knows the username, may target accounts with that username on various websites, triggering the alarm without having stolen the ciphertext. For example, the attacker may obtain a username and password from a password breach and use them on other websites to compromise the account. Additionally, an attacker may attempt to trigger false alarms for DoS attacks. To mitigate these false alarms, we recommend using random usernames and passwords for both real and honey accounts. Some password vault applications, such as Bitwarden, already provide random usernames. Since the random usernames are only known to the attacker who has stolen the ciphertext, other attackers cannot trigger false alarms.

Table 2: Time cost of our new honey vault scheme

PW number[a]	**10**	**20**	**50**
Encode the whole vault (s)	0.64	1.23	3.08
Decode the whole vault (s)	0.47	0.94	2.35
Decode a password (s)	0.24	0.47	1.16
Update/add/encode a password (s)	0.06	0.06	0.06

[a] The count includes only human-generated passwords. In contrast, randomly generated passwords require only $O(1)$ time for encoding and decoding, regardless of the total number of passwords.

In the experiments, we consider two cases for honey accounts: 1) Ideal Case, where the attacker cannot distinguish between real and honey accounts; 2) Practical Case, where the attacker can directly identify 50% of real accounts using side information, such as if the website provides login notices. Naturally, in the Practical Case, the attacker should first attempt to log in to the definitely real accounts, followed by the remaining ones, which are a mix of real and honey accounts.

Honey accounts are commonly used in authentication for leakage detection [8]. Cheng et al. [6] proposed the use of honey accounts in honey vaults, but did not implement the idea. We are the first to implement this mechanism and evaluate its security.

5.3 Efficiency

We implement our honey vault scheme and evaluate its efficiency. Although the cost of the vault is not trivial, it remains feasible for most use cases on typical PCs and smartphones.

The full model has 85.2 million parameters, stored with 32-bit precision, requiring 326MB of storage in Open Neural Network Exchange (ONNX) format. If stored with 16-bit precision, the storage and memory cost are reduced by about half. Additionally, techniques such as model quantization [31] could further decrease the cost.

We evaluate the efficiency of our full model on a desktop with an NVIDIA 3090 GPU, requiring 845MB of memory using ONNX Runtime. Table 2 presents the time cost of our honey vault scheme with the full model. The primary cost is attributed to encoding and decoding operations, while the cost for symmetric encryption is negligible (around 0.01 s). The time cost increases with vault size: For a vault size of 20, encoding and decoding take 1.23 s and 0.94 s, respectively. For a vault size of 50, these times increase to 3.08 s and 2.35 s. We also conduct evaluations on a Kirin 888 chipset smartphone, which generally requires 3.5 times the time.

Performance degradation with increasing vault size is due to the $O(l^2)$ time complexity of the Transformer for an l-length sequence. As shown in Fig. 4, in our honey vault scheme, l is the sum length of human-generated passwords.

For real-world applications, we propose two strategies to reduce time cost: 1) Place the most frequently used passwords at the beginning of the password sequence. This en-

sures that, in most cases, only the initial part of the sequence needs to be decoded, as users tend to access a small subset of their accounts frequently. 2) Encourage users to use random passwords where possible. Encoding random passwords is straightforward and independent of other passwords, requiring only $O(1)$ complexity without the use of the Transformer. Minimizing the number of human-generated passwords can enhance both security and efficiency.

6 Security Evaluation

We evaluate our and existing honey vault scheme through both traditional evaluations under distinguishing attacks and our practical evaluations under online guessing.

6.1 Experimental Setting

We construct a dataset of 63 million vaults and randomly select 7 million vaults as the training set (real vaults) for both the vault model and the attacks. We limit the training set size to this number because expanding it requires significantly more computational resources without greatly affecting the evaluation outcomes. For the test set, we randomly select 2,000 vaults from the remaining dataset. Based on the law of large numbers and our experiments, increasing the test set size does not substantially impact the results.

To train certain attacks, such as binary classifiers, decoy samples are needed. We sample 7 million vaults from the vault models, matching the number of real vaults used.

To evaluate the security of honey vault schemes under attacks, we generate decoy vaults for the real vaults in the test set according to each scheme. For traditional evaluation, following previous works [6, 7, 11, 12], we generate $N-1$ decoy vaults for each real vault, where $N = 1,000$.

For practical evaluation, we generate $N = 100,000$ vaults for each real vault to simulate offline decryption using a dictionary of the top N master passwords. According to Eq. (3), the attacker should obtain one vault for each master password and sum the results. However, generating vaults for all master passwords requires exponential time complexity. Our setting of a limited N aligns with the behavior of real-world attackers. Besides, our experiments show that increasing N has little effect on the ranking of account passwords within 1,000 online guesses for Cheng et al.'s [6] and our scheme. This is likely because items in Eq. (3) with master passwords ranked higher than N have relatively low probabilities and thus little influence on the sum.

With $N = 100,000$ decoy vaults and a real vault, we first sample a master password from the master password distribution and assign it to the real vault. Then, we assign each of the first N master passwords to a decoy vault, unless it has already been assigned to the real vault. This simulation mimics the decryption of N vaults using the top N master passwords, which are then given to the attacker along with the corresponding master passwords.

Table 3: The accuracy α or AUC of distinguishing attacks against honey vault schemes.

Attack\Scheme	Golla [12]	Cheng [6]	Our
DeBERTaBase [14]	95.79%	83.75%	64.35%
DPCNN [17]	95.05%	78.52%	57.57%
FastText [18]	89.98%	67.25%	53.42%
TextCNN [20]	94.15%	75.32%	54.75%
SinglePwAttack [6]	66.47%	44.52%	56.89%
PwSimilarityAttack [6]	63.29%	52.07%	51.20%
HybridAttack [6]	70.24%	45.35%	57.61%
KLAttack [12]	33.39%	66.32%	52.16%

To evaluate the performance of our two measures, we simulate Measure II by generating honey accounts, assuming the user manually creates 10 such accounts. If the attacker attempts to log in to honey accounts, we assume the user is notified and changes all account passwords. Although the attacker may continue making online guesses, they will not succeed. As discussed in Section 5.2, we consider two cases regarding the distinguishability of honey vaults: 1) Ideal Case, where the attacker cannot distinguish between real and honey accounts; 2) Practical Case, where the attacker can directly identify 50% of real accounts using side information.

We compare our scheme with Golla et al.'s static honey vault scheme [12] and Cheng et al.'s scheme [6]. We exclude Chatterjee et al.'s scheme [4] due to its weaker security compared to both Golla et al.'s [12] and Cheng et al.'s [6] schemes, as well as its inability to utilize Cheng et al.'s IS-PMTE [7] to resist encoding attacks, owing to its high ambiguity and complexity. In contrast, Golla et al.'s scheme can use IS-PMTE by pruning unnecessary breaches to reduce ambiguity, as described in [7]. We also exclude Golla et al.'s adaptive scheme, which has been proven insecure both experimentally and theoretically.

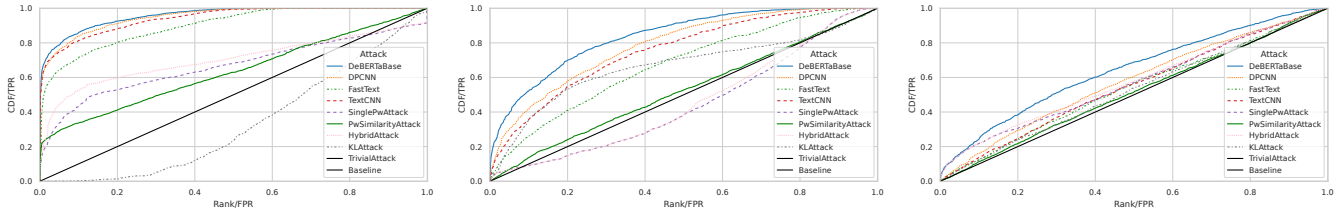
6.2 Traditional Evaluation against Distinguishing Attacks

Security Metrics. As discussed in Section 4.6, the traditional evaluation uses RCDF and distinguishing accuracy as metrics, which are essentially equivalent to ROC and AUC.

Experimental Results. As shown in Fig. 5 and Table 3, our attack based on DeBERTa achieves the highest accuracy: 95.79%, 83.75%, and 64.35% against the schemes of Golla et al., Cheng et al., and ours, respectively. Overall, our attacks using text classifiers outperform previous methods, with the pre-trained model, DeBERTa, performing the best.

In addition, our honey vault scheme demonstrates stronger security than the prior ones: no attack exceeds an accuracy of *64.35%*, which is close to the baseline accuracy of 50% from a trivial attack.

Discussion. Previous attacks and vault models typically



(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 5: Honey vault schemes in traditional evaluation under distinguishing attacks (ROC or RCDF).

Table 4: Average number of cracked accounts in honey vaults without measures under 1,000 online attempts

Attack\Scheme	Golla [12]	Cheng [6]	Our
DeBERTaBase [14]	0.7608	0.6517	0.5080
DPCNN [17]	0.7543	0.5516	0.5055
FastText [18]	0.5861	0.5030	0.5045
TextCNN [20]	0.6431	0.5065	0.5025
SinglePwAttack [6]	0.3148	0.2928	0.3173
PwSimilarityAttack [6]	0.5270	0.4919	0.5030
HybridAttack [6]	0.3514	0.2978	0.3188
KLAttack [12]	0.3849	0.4705	0.4895
TrivialAttack	0.4319	0.4920	0.5035

rely on traditional statistical learning methods and adopt simple designs for password similarity. For example, Cheng et al.'s attack uses a Bayes classifier with only two binary features, which fails to capture the complexity of real-world password reuse habits. This limitation partly stems from the lack of large-scale datasets in earlier studies, which makes it difficult to train more advanced models.

We construct a large dataset that enables the use of advanced deep learning models for both honey vault design and security evaluation. Our results show that a general-purpose text classifier outperforms existing distinguishing attacks specifically designed for honey vaults, and pre-trained models offer additional advantages. Moreover, our Transformer-based model provides better representation and characterization of password vaults when trained on large datasets. We believe that advances in NLP will continue to benefit the design and evaluation of honey vault schemes.

6.3 Practical Evaluation against Online Guessing

Security Metrics. In our practical evaluation, we use the number of cracked accounts y versus the number of online login attempts x as the metric.

Experimental Results. Our DeBERTaBase attack and vault scheme achieve the best performance, as shown in Fig. 6–14 and Table 7 (most results are provided in the Appendix).

Fig. 6 and Table 4 show the security of honey vaults without any measures. Our attacks based on text classifiers outperform prior methods, so we omit the results of previous attacks in the remaining cases. Although our vault scheme significantly outperforms existing schemes under traditional evaluation metrics, it shows only marginal improvement under online guessing attacks. With 1,000 online attempts by DeBERTaBase, the attacker can compromise 0.7608, 0.6517 and 0.5080 accounts on average under the schemes of Golla et al. [12], Cheng et al. [6], and ours, respectively. Moreover, the DeBERTaBase attack performs similarly to a trivial attack when targeting our scheme.

Table 5: Average number of cracked accounts in honey vaults with 1,000 online attempts by the best attack (DeBERTaBase)

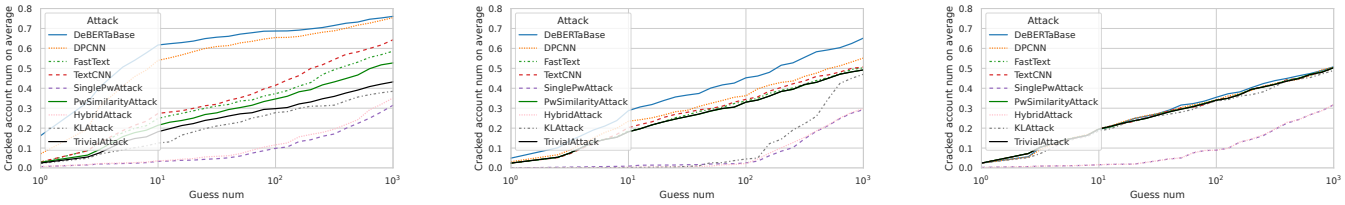
Measures and/or strong MPW[a]	Golla [12]	Cheng [6]	Ours
None	0.7608	0.6517	0.5080
M1	0.6471	0.3734	0.3108
M2 (Ideal case)	0.1196	0.0581	0.0165
M2 (Practical case)	0.3819	0.3674	0.2427
MPW	0.5375	0.4214	0.2728
M1 & M2 (Ideal Case)	0.0961	0.0611	0.0155
M1 & M2 (Practical Case)	0.3418	0.2538	0.1086
M1 & M2 (Ideal Case) & MPW	0.0736	0.0360	0.0075
M1 & M2 (Practical Case) & MPW	0.2497	0.1677	0.0546

[a] M1: Measure I; M2: Measure II; MPW: strong master password. "Ideal case" assumes the attacker cannot distinguish real from honey accounts; "Practical case" assumes the attacker can identify 50% of real accounts using side information.

Table 5 summarizes the results under the DeBERTaBase attack with different measures. Measure I significantly reduces the number of compromised accounts to 0.6471, 0.3734 and 0.3108. Measure II also substantially improves security. In the Ideal Case, where the attacker cannot distinguish real from honey accounts, the average number of cracked accounts drops to 0.1196, 0.0581 and 0.0165. In the Practical Case, where the attacker can identify 50% of real accounts using side information, the numbers reduce to 0.3819, 0.3674 and 0.2427, respectively.

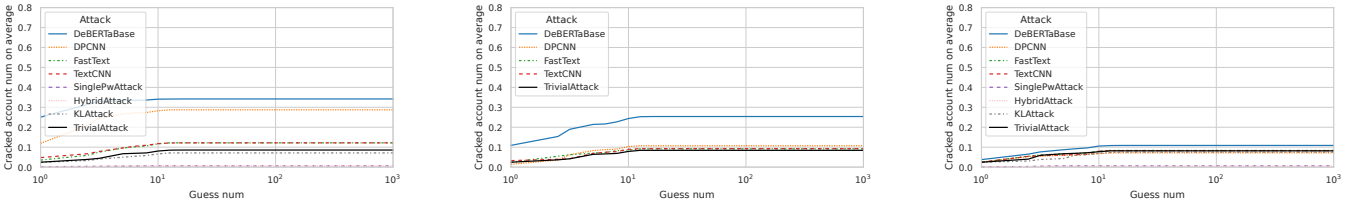
Fig. 7 illustrates the security improvement under the Practical Case with both Measure I and Measure II applied. With 1,000 online attempts, the DeBERTaBase attacker can compromise only 0.3418, 0.2538 and 0.1086 accounts across the three schemes.

When further combined with strong master passwords, the



(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 6: Honey vault schemes without measures



(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 7: Honey vault schemes with Measures I and II in the Practical Case

numbers decrease to 0.2497, 0.1677 and 0.0546.

Discussion. We present the first evaluation of the practical security of honey vaults under online guessing attacks.

Unlike in traditional evaluations, most existing attacks—except the password-similarity attack—perform worse than the trivial attack in the online guessing setting. This is because the priority scores used in these attacks are disproportionately large for some vaults. While such scores preserve the ranking order, they introduce significant errors when estimating the cumulative probability in Eq. (3). In contrast, our attacks based on binary classification, as well as the password-similarity attack using a Bayes binary classifier, exhibit strong robustness under online guessing.

Although our attacks, particularly DeBERTaBase, achieve high accuracy in distinguishing real and decoy vaults for the schemes of Golla et al. [12] and Cheng et al. [6], the number of compromised accounts remains on the same order as that of the trivial attack. For our new vault scheme, all of our attacks perform nearly identically to the trivial baseline, suggesting that the discrepancy between our vault model and the true vault distribution has little impact on practical security under online guessing. Existing vault designs already generate decoy vaults that, while not perfectly indistinguishable from the real ones, are sufficiently similar to offer attackers little additional advantage in online guessing.

We conjecture that even with more advanced attack strategies or vault models—despite leveraging sophisticated techniques and larger datasets—the number of compromised accounts may not change significantly. We argue that the relationship between attack and vault model is not a cat-and-mouse game. As the vault model approaches the real distribution, even an optimal attacker with precise knowledge of the true distribution gains diminishing advantage.

Rather than focusing solely on improving vault models, we find that two simple defense measures offer substantial security gains. Measure I not only reduces the number of compromised accounts under a fixed number of online attempts, but also limits the total number of online attempts, preventing attackers from compromising all accounts through exhaustive guessing. Measure II, by incorporating honey accounts, enables effective leakage detection and allows timely user notifications to prevent further damage. We also show that using stronger master passwords provides additional, though modest, improvements in security.

7 Conclusion

We construct a large vault dataset and leverage advanced machine learning techniques in honey vaults for both decoy generation and identification. We use various advanced text classification algorithms (especially pre-trained models) to implement distinguishing attacks, achieving clear advantages over existing vault schemes in traditional security evaluation. We further use a Transformer model to build a vault model and further a new honey vault scheme, which can effective against distinguishing attacks in traditional evaluation.

We further assess the practical security of honey vaults against online guessing. Our new scheme achieves the best performance. With 1,000 online guessing, only 0.51 accounts is cracked on average. By applying two simple measures, we enhance the scheme: 1) avoiding the encryption of passwords for websites with unlimited login attempts, and 2) using honey accounts for leakage detection. These measures reduce the cracked number to 0.11, improving the security to a practical

level. We also provide new insights. Such as, improving the probability model alone, without our measures, yields diminishing returns in practical security; even a poor model can still achieve notable practical security by using our measures. Our insights may be useful to guide the study direction on honey vaults, and promote the usage of honey vaults in the real world.

8 Acknowledge

This research is supported by National Natural Science Foundation of China (No. 62202012 and No. 62072010), BIGC Project Ea202515 and Publishing Think Tank Platform Development Project KYCPT202514.

9 Ethical Considerations

We collected the RMDC dataset via BT after it was publicly shared following the shutdown of RaidForums. Although the dataset was originally stolen by hackers and sold on RaidForums, we did not pay for it and explicitly do not support any form of criminal or unethical activity. Given the widespread public availability of RMDC via BT, we use it in our study with the goal of improving password protection techniques, rather than causing any further harm. To prevent the data from further dissemination, we have implemented the following measures for its usage and storage: 1) The raw dataset has been deleted on the server after cleaning. 2) The cleaned dataset is securely stored on our servers and cannot be downloaded locally. 3) Access is restricted to participants of this work, who must authenticate using certificates. 4) Participants are only permitted to view a limited number of items in the dataset during the cleaning process. 5) The cleaned dataset does not contains no usernames or directly identifiable information.

10 Open Science

To ensure reproducibility while safeguarding user privacy, we have open-sourced our code and adopted a strict verification process for dataset access. The source code is publicly available at https://doi.org/10.5281/zenodo.15612142, while the dataset is hosted with restricted access and encryption at https://doi.org/10.5281/zenodo.15646752. Access to the dataset is granted only after verifying the requester's identity and academic credentials. Additionally, recipients must agree to enforce appropriate security measures for storing and handling the data. Importantly, the shared dataset has been sanitized to remove usernames and other sensitive information; it does not include the original raw data.

References

[1] 1Password. Turn on two-factor authentication for your 1password account. https://support.1password.com/two-factor-authentication/.

[2] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-resistant password management. In *ESORICS 2010*, pages 286–302. Springer.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NIPS 2020*, volume 33, pages 1877–1901.

[4] Rahul Chatterjee, Joseph Bonneau, Ari Juels, and Thomas Ristenpart. Cracking-resistant password vaults using natural language encoders. In *IEEE S&P 2015*, pages 481–498, 2015.

[5] Yu Chen, Yang Yu, and Lidong Zhai. InfinityGauntlet: Expose smartphone fingerprint authentication to brute-force attack. In *USENIX Security 2023*, pages 2027–2041, 2023.

[6] Haibo Cheng, Wenting Li, Ping Wang, Chao-Hsien Chu, and Kaitai Liang. Incrementally updateable honey password vaults. In *USENIX Security 2021*, pages 857–874, 2021.

[7] Haibo Cheng, Zhixiong Zheng, Wenting Li, Ping Wang, and Chao-Hsien Chu. Probability model transforming encoders against encoding attacks. In *USENIX Security 2019*, pages 1573–1590, 2019.

[8] Forbes Councils. Unlocking the future of identity protection: How honeytokens and honey accounts redefine cybersecurity, Nov 2023. https://www.forbes.com/councils/forbestechcouncil/2023/11/17/unlocking-the-future-of-identity-protection-how-honeytokens-and-honey-accounts-redefine-cybersecurity/.

[9] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS 2014*, 2014.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In

Jill Burstein, Christy Doran, and Thamar Solorio, editors, *NAACL 2019*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[11] Fei Duan, Ding Wang, and Chunfu Jia. A security analysis of honey vaults. In *IEEE S&P 2024*, pages 1424–1442, 2024.

[12] Maximilian Golla, Benedict Beuscher, and Markus Dürmuth. On the security of cracking-resistant password vaults. In *ACM CCS 2016*, pages 1230–1241, 2016.

[13] Amber Gott. LastPass security notification, March 2017. https://blog.lastpass.com/2017/03/important-security-updates-for-our-users.html/.

[14] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2021.

[15] Zhicong Huang, Erman Ayday, Jacques Fellay, Jean-Pierre Hubaux, and Ari Juels. Genoguard: Protecting genomic data against brute-force attacks. In *IEEE S&P 2015*, pages 447–462, 2015.

[16] Joseph Jaeger, Thomas Ristenpart, and Qiang Tang. Honey encryption beyond message recovery security. In *EUROCRYPT 2016*, pages 758–788, 2016.

[17] Rie Johnson and Tong Zhang. Deep pyramid convolutional neural networks for text categorization.

[18] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *EACL 2017*.

[19] Ari Juels and Thomas Ristenpart. Honey encryption: Security beyond the brute-force bound. In *EUROCRYPT 2014*, pages 293–310. Springer, 2014.

[20] Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *EMNLP 2014*.

[21] LastPass. What is 2fa & how two-factor authentication works. https://www.lastpass.com/solutions/authentication/two-factor-authentication.

[22] LastPass. 12-22-2022: Notice of security incident, 2022. https://blog.lastpass.com/posts/notice-of-recent-security-incident.

[23] Sanam Ghorbani Lyastani, Michael Schilling, Sascha Fahl, Sven Bugiel, and Michael Backes. Better managed than memorized? studying the impact of managers on password strength and reuse. In *USENIX Security 2018*, pages 203–220, 2018.

[24] Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. A study of probabilistic password models. In *IEEE S&P 2014*, pages 538–552, 2014.

[25] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *IEEE S&P 2019*, pages 814–831, 2019.

[26] Dario Pasquini, Giuseppe Ateniese, and Carmela Troncoso. Universal neural-cracking-machines: Self-configurable password models from auxiliary data. In *IEEE S&P 2024*, pages 1365–1384, Los Alamitos, CA, USA, May 2024. IEEE Computer Society.

[27] Dario Pasquini, Ankit Gangwal, Giuseppe Ateniese, Massimo Bernaschi, and Mauro Conti. Improving password guessing via representation learning. In *IEEE S&P 2021*, pages 265–282, 2021.

[28] Sarah Pearman, Shikun Aerin Zhang, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Why people (don't) use password managers effectively. In *SOUPS 2019*, pages 319–338, 2019.

[29] Security.org. 2024 password manager industry report and statistics, Dec 2024. https://www.security.org/digital-safety/password-manager-annual-report/.

[30] Joe Siegrist. LastPass security notification, May 2011. https://blog.lastpass.com/2011/05/lastpass-security-notification.html/.

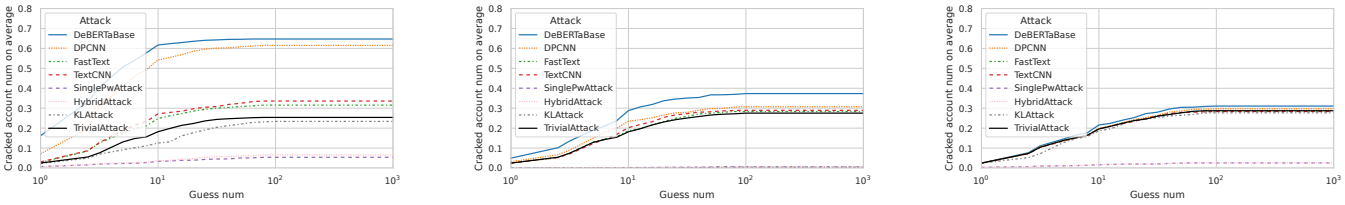
[31] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6), December 2022.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS 2017*, volume 30, 2017.

[33] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *CCS 2016*, pages 1242–1254, 2016.

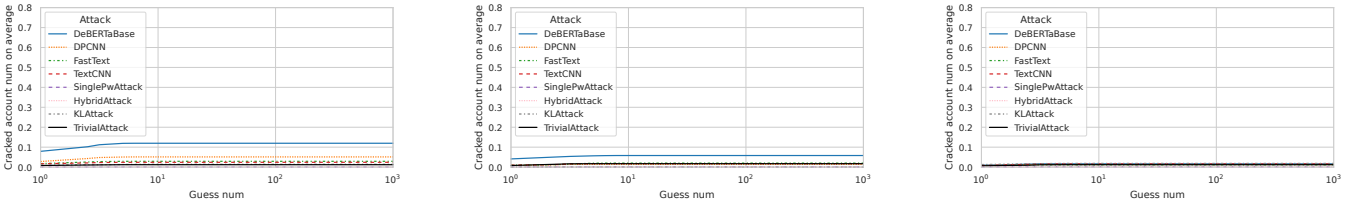
[34] Wikipedia. Raidforums. https://en.wikipedia.org/wiki/RaidForums.

[35] Davey Winder. Lastpass hackers allegedly stole $5 million this week, report, 2024. https://www.forbes.com/sites/daveywinder/2024/12/18/lastpass-hackers-allegedly-stole-5-million-this-week-report/.



(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 8: Honey vault schemes with Measure I



(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

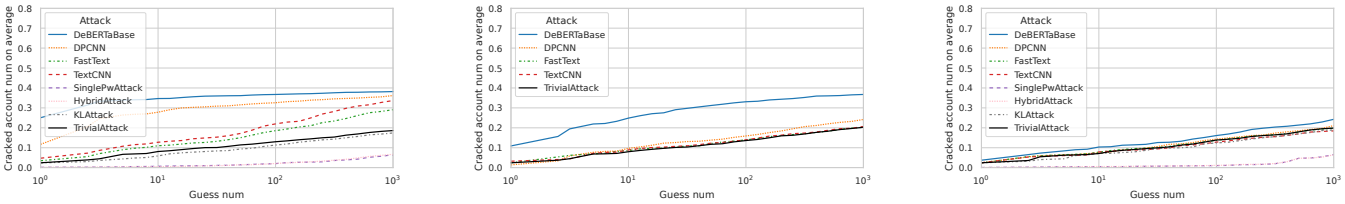
Figure 9: Honey vault schemes with Measure II in the Ideal Case

Table 6: The 28 datasets in RMDC that were used and cleaned

Website	Original	Cleaned	Ctry	Category
000Webhost	15,283,553	14,782,714	EU	Hosting
17173	18,333,811	17,411,727	CN	Game
brazzers	928,071	925,408	US	Adult
CSDN	6,428,631	6,425,841	CN	Forum
Dodonew	16,283,139	15,915,585	CN	Cashier
Ispeak	8,044,382	3,864,068	CN	Game
Neopets	68,743,268	34,891,498	US	Game
NetEase	288,584,536	272,583,347	CN	Email
Pokemoncreed	139,630	139,606	JP	Game
Tianya	31,250,552	17,706,677	CN	Forum
Ys168	328,586	325,497	CN	Service
7k7k	14,928,048	7,528,441	CN	Game
Badoo	25,889,161	25,353,232	RU	Social
Clixsense	2,222,507	2,192,738	US	Survey
Couponmom	10,855,708	10,685,769	US	Coupons
Crackingforum	277,351	268,141	US	Forum
Imgur	1,755,559	1,713,501	US	Social
Lookbook	870,365	814,237	US	Social
Mail.ru	3,736,301	3,725,945	RU	Social
Mate1	27,403,958	26,507,803	US	Social
Mop	2,167,944	1,876,799	CN	Game
Renren	4,768,599	4,589,890	CN	Social
Taobao	14,979,981	14,955,315	CN	Shop
Twitter	41,001,171	24,999,366	US	Social
VK	92,473,648	81,019,315	RU	Social
Yahoo voice	259,851	251,501	US	Voice
Warmane	494,389	480,601	US	Game
Yahoo	456,580	433,969	US	Search

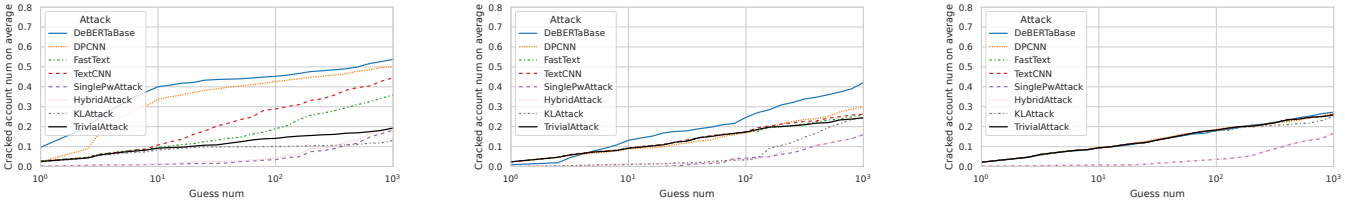
Table 7: Average number of cracked accounts in honey vaults with 1,000 online attempts

Attack\Scheme	Golla et al. [12]	Cheng et al. [6]	Our
Measures I			
DeBERTaBase	0.6471	0.3734	0.3108
DPCNN	0.6151	0.3073	0.2968
FastText	0.3153	0.2833	0.2858
TextCNN	0.3358	0.2893	0.2833
TrivialAttack	0.2538	0.2753	0.2873
Measures II in the Ideal Case			
DeBERTaBase	0.1196	0.0581	0.0165
DPCNN	0.0511	0.0135	0.0130
FastText	0.0290	0.0200	0.0115
TextCNN	0.0240	0.0175	0.0160
TrivialAttack	0.0120	0.0170	0.0120
Measures II in the Practical Case			
DeBERTaBase	0.3819	0.3674	0.2427
DPCNN	0.3619	0.2417	0.2112
FastText	0.2908	0.2027	0.1977
TextCNN	0.3368	0.2067	0.1847
TrivialAttack	0.1862	0.2027	0.1987
Strong master password			
DeBERTaBase	0.5375	0.4214	0.2728
DPCNN	0.5015	0.2988	0.2708
FastText	0.3594	0.2583	0.2513
TextCNN	0.4454	0.2628	0.2633
TrivialAttack	0.1927	0.2447	0.2578
Measures I & II in the Ideal Case			
DeBERTaBase	0.0961	0.0611	0.0155
DPCNN	0.0596	0.0180	0.0120
FastText	0.0230	0.0260	0.0115
TextCNN	0.0275	0.0115	0.0120
TrivialAttack	0.0055	0.0195	0.0060
Measures I & II in the Practical Case			
DeBERTaBase	0.3418	0.2538	0.1086
DPCNN	0.2873	0.1071	0.0716
FastText	0.1216	0.0916	0.0836
TextCNN	0.1216	0.0926	0.0826
TrivialAttack	0.0856	0.0841	0.0816
Measures I & II in the Ideal Case & Strong master password			
DeBERTaBase	0.0736	0.0360	0.0075
DPCNN	0.0325	0.0050	0.0065
FastText	0.0165	0.0270	0.0055
TextCNN	0.0125	0.0265	0.0070
TrivialAttack	0.0100	0.0255	0.0055
Measures I & II in the Practical Case & Strong master password			
DeBERTaBase	0.2497	0.1677	0.0546
DPCNN	0.1857	0.0495	0.0506
FastText	0.0636	0.0495	0.0470
TextCNN	0.0946	0.0495	0.0480
TrivialAttack	0.0511	0.0485	0.0506



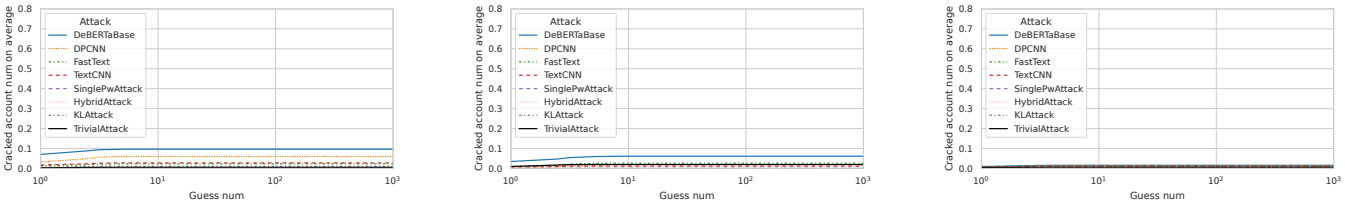
(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 10: Honey vault schemes with Measure II in the Practical Case



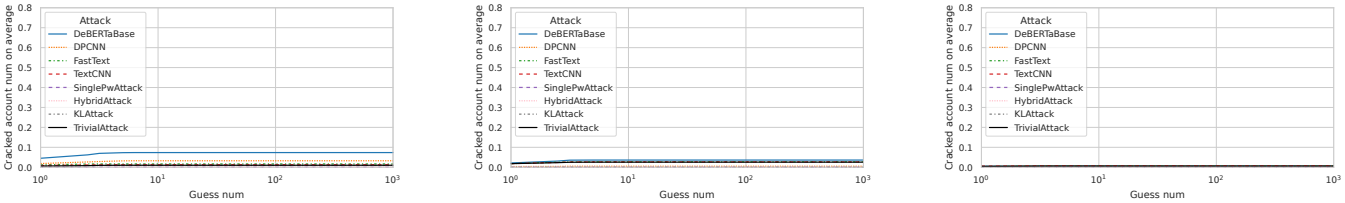
(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 11: Honey vault schemes with strong master passwords



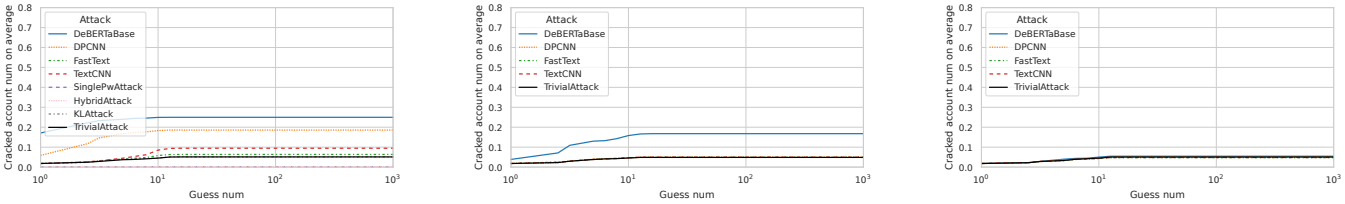
(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 12: Honey vault schemes with Measures I and II in the Ideal Case



(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 13: Honey vault schemes with Measures I and II and strong master passwords in the Ideal Case



(a) Golla et al.'s scheme [12] (b) Cheng et al.'s scheme [6] (c) Our scheme

Figure 14: Honey vault schemes with Measures I and II and strong master passwords in the Practical Case