Incrementally Updateable Honey Password Vaults

Haibo Cheng[1], Wenting Li[1], Ping Wang[1], Chao-Hsien Chu[2], Kaitai Liang[3]
[1]*Peking University* [2]*Pennsylvania State University* [3]*Delft University of Technology*

Abstract

Password vault applications allow a user to store multiple passwords in a vault and choose a master password to encrypt the vault. In practice, attackers may steal the storage file of the vault and further compromise all stored passwords by offline guessing the master password. *Honey vaults* have been proposed to address the threat. By producing plausible-looking decoy vaults for wrong master passwords, honey vaults force attackers to shift offline guessing to online verifications.

However, the existing honey vault schemes all suffer from intersection attacks in the multi-leakage case where an old version of the storage file (e.g., a backup) is stolen along with the current version. The attacker can offline identify the decoys and completely break the schemes. We design a generic construction based on a *multi-similar-password model* and further propose an *incremental update* mechanism. With our mechanism, the attacker cannot get any extra advantages from the old storage, and therefore degenerates to an attacker only with knowledge of the current version.

To further evaluate the security in the traditional single-leakage case where only the current version is stolen, we investigate the theoretically optimal strategy for online verifications, and propose practical attacks. Targeting the existing schemes, our attacks crack 33%–55% of real vaults via *only one-time* online guess and achieve 85%–94% accuracy in distinguishing real vaults from decoys. In contrast, our design reduces the values of the two metrics to 2% and 58% (close to the ideal values 0% and 50%), respectively. This indicates that the attackers needs to carry out *2.8x–7.5x* online verifications to break our scheme. Since online verifications can be quickly detected and prevented, our design achieves a significant improvement on security.

1 Introduction

Password vaults, a.k.a *wallets* or *managers*, are highly recommended for password management. A user can store multiple passwords in a vault and further set a master password to encrypt the vault. The user thus only needs to remember the master password instead of a long list of daily-use passwords. In practice, the user usually uses the vault among multiple clients (e.g., smartphone or PC), and requires its synchronization via online services. The synchronization may be provided by the vault applications (e.g., LastPass and 1Password) or third-party file sync services (e.g., Dropbox and iCloud). However, the sync services may suffer from leakage [27, 32, 44, 45, 48], which leads to a great threat for password vaults.

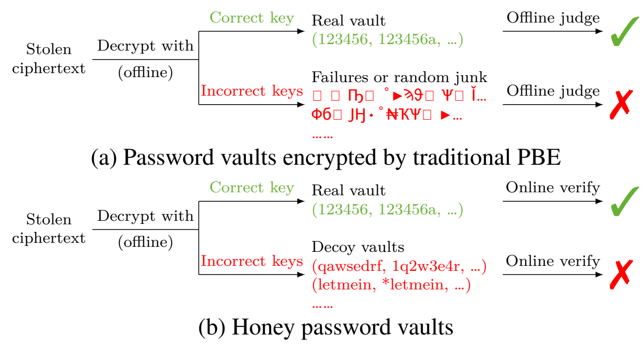


Figure 1: The difference between traditional and honey password vaults in the view of attackers.

If an attacker steals the storage file of a vault (including the ciphertext), the attacker can launch guessing attacks against the master password to compromise all stored passwords. For a vault encrypted by traditional password-based encryption (PBE), decrypting it with an incorrect guess will yield a failure (i.e., $\perp$) or random junk. So the attacker can immediately identify the validity of guesses *offline*. In addition, since the master password is human-memorable, it may be low-entropy [11, 53] and could be guessed as easily as a website login password [36, 51, 52]. Accordingly, the attacker can efficiently carry out this offline attack with a high probability of success.

Honey password vault [9] is proposed to address this threat. Its core idea is to *generate plausible-looking decoy vaults for incorrect guesses to confuse attackers*. As shown in Fig. 1, launching offline guessing produces many decoy vaults (with a real one), which need to be online verified (i.e., trying to log in with passwords in the vaults). By pushing attackers to online verification, the honey vault mechanism significantly enhances the security of vaults, as the verification can be practically detected and prevented [17, 21, 42].

The design of decoy vaults originates from Bojinov et al. [9]. Their proposed *Kamouflage* pre-generates a *static amount* (e.g., 1,000) of decoy vaults with corresponding decoy master passwords, and further stores them with the real ones. Later, Chatterjee et al. [13] introduced a honey vault scheme *NoCrack* based on *Honey Encryption (HE)* [23]. HE is used to turn a vault to a random-looking bit string called *seed* with a probabilistic encoder—*distribution transforming encoder*—and further encrypt the seed. Due to the "honey" feature provided by HE, using an *arbitrary* wrong master password in decryption can yield a random-looking seed that will be further decoded to a decoy vault on the fly. This brings attackers much more difficulties to tell the real vault, compared with the pre-generating method. Subsequently, Golla et al. [18] proposed *adaptive encoders* which adjust themselves according to the encrypted vault to make decoys more similar to it. Cheng et al. [14] found both Chatterjee et al.'s [13] and Golla et al.'s [18] encoders suffer from *encoding attacks*. They further proposed a generic transformation that can convert a probability model to an encoder resisting encoding attacks.

However, all existing honey vault schemes suffer from *intersection attacks* in the multi-leakage case where an old version of the storage file is stolen along with the current version. This is an open question left in [13, 18]. More specifically, the schemes only provide full update for a vault, i.e., reprocessing the updated vault as a brand new one, even if a user just changes a password (or add a new one). This yields a totally different new version for each decoy vault (by decrypting the new ciphertext with the same master password); and meanwhile the old and new versions of the real vault are the same except for the changed password. Hence, the attacker can offline identify the real vault, according to the similarity between the new and old versions of the vaults. This is a *realistic* threat because: 1) the old version of the storage file usually is backed up and stored with the current version by the online services or applications, (for example, Dropbox keeps all history versions of files for 30 days [16], and 1Password automatically creates a backup for each change [7]); 2) the online storage may suffer from multiple leakages due to increasing number of network attacks and software bugs [6, 20, 26, 34, 44–46].

1.1 Our Contributions

To resist intersection attacks, we propose a generic construction and an *incremental update* mechanism for HE-based honey vaults. We build our construction from: 1) a *multi-similar-password model* which models the conditional password distribution given multiple old passwords (i.e., the old vault); and 2) the corresponding conditional encoder which can encode a password given multiple old ones. With the construction, we can encode the changed (or added) password to a (sub) seed and pad it to the tail of the vault seed. With a prefix-keeping PBE scheme, the similarity between the old and new versions of each decoy vault is kept the same as that for the real vault. Therefore, our scheme resists intersection attacks. Formally, the attacker cannot get any extra advantages from the old storage file, and degenerates to an attacker in the single-leakage case (where only the current version is stolen).

To further evaluate the security of (HE-based) honey vault schemes against distinguishing attacks in the single-leakage case, we formally investigate the optimal strategy for online verifications and further propose several practical attacks[1]. For the existing schemes [13, 14, 18], our attacks crack 33%–55% of real vaults via *only one-time* online guess and achieve 85%–94% accuracy in distinguishing real vaults from decoys. We further find that the adaptive encoder proposed by Golla et al. [18] does leak extra information about the real vault. This makes it more insecure than its static variant. Leveraging the leaked information, our attacks can achieve 91%–93% distinguishing accuracy against the adaptive encoder, which is 6.2%–9.0% higher than that of the static variant.

To instantiate our construction, we design a multi-similar-password model according to users' password-generating habits. The design is built on the top of a single-password model (capturing how the user creates a brand new password), a single-similar-password model (capturing how a user creates a new password by reusing an old one) and an unreused probability function. For our design, the existing and our proposed attacks only crack at most 2% of real vaults via one-time online guess and achieve at most 58% distinguishing accuracy. The results are close to 0% and 50% maintained by an ideal secure scheme, respectively. This also means the attacker has to carry out *2.8x–7.5x* online verifications against our scheme as compared to others. Since online verifications can be quickly detected and prevented [17, 21, 42], our design achieves a significant improvement on security.

In summary, we describe our main contributions as follows.

1. We propose a new generic construction and an incremental update mechanism for HE-based honey vault schemes, which resists intersection attacks.
2. We formally investigate the optimal strategy for online verifications and further propose several practical attacks, which can effectively distinguish real and decoy vaults for the existing honey vault schemes.
3. We instantiate our construction with a well-designed multi-similar-password model, which can generate more plausible-looking decoys.

[1]Here, we only focus on human-generated passwords. Although many vault applications recommend users to use randomly-generated passwords, they always store some human-generated passwords in the vaults [35, 40] (note these passwords may inherit from the old password management methods or be used for the convenience of manual entry on gaming consoles). For these passwords, it is of great challenge to generate indistinguishable decoys [13] (it is trivial for randomly-generated ones).

2 Background and Related Work

2.1 Traditional Solutions to Offline Guessing

A straightforward solution to master password guessing is to leverage a special password hashing as the key derivation function (KDF) used in password-based encryption (PBE), such as an iterated hash function [25, 43], a memory-hard function [10, 41]. LastPass employs this solution (using the 100,100 rounds of PBKDF2-SHA256 [4]) to increase the computational cost of attackers in launching master password guess. Nevertheless, the cost of a valid user is also increased by the same factor. Without leveraging heavy hashing, one may use an extra key stored on a device (e.g., iOS keychain [5], a server [29]), to enhance the master password to a cryptographic key for further encryption, like 1Password [5]. But this has a defect that if the device gets lost without any backup, all the passwords stored in the vault cannot be recovered anymore. Note that these solutions can be used in honey vault schemes to achieve complementary protection. There may be other approaches but we don't explore them here. We will only focus on the solutions based on honey vaults.

2.2 Honey Encryption

Honey Encryption (HE) [23], proposed by Juels and Ristenpart, can resist the brute-force attack by yielding plausible-looking messages for arbitrary incorrect keys, even in the case where a low-entropy key (e.g., password) is used. Later, Jaeger et al. [22] proved that HE satisfies the stronger notions of target-distribution semantic security and target-distribution non-malleability. The core design of HE relies on an encoder, called *distribution transforming encoder (DTE)*, being able to capture the message distribution. Intaking a message M following some distribution $\mathcal{M}$, DTE can encode it to a bit string S called *seed* which is indistinguishable from a random string. HE further encrypts S to a ciphertext C by a traditional but carefully-chosen PBE scheme with a key K. Decrypting C with a wrong key K' (e.g., a guessing key from attackers), the carefully-chosen PBE (e.g., the CTR-mode AES with PBKDF used in [13, 18, 23]) can yield a random-looking bit string S'. DTE then decodes S' into a honey message M' which is sampled from the same distribution $\mathcal{M}$. Note in the context of honey vault, the message and key correspond to the password vault and master password, respectively.

Juels and Ristenpart use *inverse sampling* to convert a distribution to an encoder called *IS-DTE*. This method performs well for simple distributions, e.g., uniform distributions. But when handling messages (e.g., natural language) with a huge space size and a complex distribution, it definitely yields explosive complexity in time and storage space. To tackle this problem, Chatterjee et al. [13] introduce a *natural language encoder*, and later Cheng et al. [14] propose a *probability model transforming encoder* (see Sections 2.3 and 2.4).

Table 1: The storage format of honey vaults

Plaintext part	Domain	Facebook	Myspace	000Webhost	Twitter	...
	Username	Aaron	Aaron1	AaronJ	Aaron	...
	Randomly-generated	No	No	Yes	No	...
	Password position	1	3	1	2	...

Ciphertext part	Human-generated	(123456, 123456a, 1234567, ...)
	Randomly-generated	(cYp97@v84G$9GN$\hat{v}$s%3R, ...)

Note: Each randomly-generated password is encoded by the encoder for the uniform distribution and further encrypted separately. All human-generated passwords are encoded by the encoder for the vault model and further encrypted as a whole.

2.3 HE-based Honey Vault Schemes

Unlike traditional solutions to offline master password guessing, honey vault schemes yield decoy vaults for incorrect guesses and therefore force attackers to online verify these decoy vaults. We only focus on HE-based schemes [13, 14, 18] in this paper due to their advantage on security.

Storage format. The existing schemes only use HE to encrypt passwords and leave other parts (e.g., domains and usernames) in plaintext[2]. We give an example in Table 1 to show the storage format.

Vault model. The probability model for password vaults (vault model, for short) is the foundation used to generate indistinguishable decoys. It should characterize the real vault distribution as precisely as possible. Because of users' various password generation (and reuse) habits, it is a great challenge to design models for human-generated passwords (note it is trivial for randomly-generated passwords [13] and so we do not consider them in this paper). The existing vault models [13, 18] choose a single-password model as the base to characterize the single-password distribution and further extend it to capture the similarity (reuse habits) among multiple passwords in a vault.

Chatterjee et al. [13] use the probabilistic context-free grammar (PCFG) model and extend it by the sub-grammar approach. PCFG is first used by Weir et al. [52] (*Weir-PCFG*) in password cracking, and its basic idea is to capture the generation of password under several predefined rules. Chatterjee et al. [13] enrich the PCFG with more password generation

[2]One may choose to further encrypt domains and usernames. However, this may easily break the "honey" property of HE. Note that a real username is registered on the domain, but its decoy is not. Thus, using a decoy username on registration will always lead to a success (since the website will regard it as a "new" registration). This attack is hard to prevent, and may threaten the security of honey vault. Without using encryption, Chatterjee et al. [13] provide another way to hide domains. They generate decoy accounts for the domains where users have not registered, and further store the decoy usernames in plaintext. But attackers still can check these decoy accounts by verifying the usernames via the aforementioned registration tactic. It seems that there does not exist an effective domain-hiding solution in the context of honey vault. And providing the solution is beyond the scope of this paper.

rules and yield the *Chatterjee-PCFG*. Based on *Chatterjee-PCFG*, they define the *sub-grammar* approach that captures the generation of a vault (consisting of multiple passwords) under a small rule set. Since the rule set is relatively small (indicating that the methods of password generation are limited), passwords may share the same rules, which easily yields "password similarity".

Golla et al. [18] find that decoy vaults generated by Chatterjee et al.'s model [13] can be distinguished. To design a more secure vault model, they choose 3rd-order Markov model (denoted as *Golla-Markov*) and extend it by the reuse-rate approach. Markov model [36] is another widely used single-password model, capturing the generation of a password character by character. Based on *Golla-Markov*, the *reuse-rate* approach is designed to capture the generation of a vault with a simplification that a user has one single password and reuses it for different accounts by modifying its last i characters ($0 \leq i \leq 5$). They assume that the reuse rate (for each i) follows a Gaussian distribution for quantification.

In addition, Golla et al. introduce and apply *adaptive* concept to vault model, encoder and honey vault scheme. Before encrypting a real vault V, an adaptive scheme adjusts its vault model (as well as its encoder) according to V. With well-designed adjustments, an adaptive model may produce decoys that are more similar to V, and bring more difficulties to attackers in identifying them. Following the concept, Golla et al. [18] further present an adaptive Markov model. This model directly increases the probabilities of n-grams appeared in the real vault.

Note that Cheng et al. [14] do not propose a new vault model but leverage/recommend Golla et al.'s design.

Encoder. An encoder for a vault model should encode a vault sampled from the model to a seed being indistinguishable from a random bit string. However, the natural language encoders designed by Chatterjee et al. [13] and used by Golla et al. [18] fail to achieve this requirement, and therefore suffer from encoding attacks [14]. Cheng et al. [14] tackle this vulnerability by employing their encoders for the old models. To evaluate the existing honey vault schemes without the negative effect caused by encoding attacks, we will adopt Cheng et al.'s encoders to [13] and [18] in this paper, and still refer to the resulting schemes as Chatterjee et al.'s and Golla et al.'s. Under our adoption, Golla et al.'s scheme (with Cheng et al.'s encoder) becomes the same as Cheng et al.'s scheme (with Golla et al.'s model as they recommended).

Deployment consideration. Due to the special feature of honey vaults, if a user enters an incorrect master password (e.g., a typo), it will get a decoy vault and further lead to a login failure. Dynamic security skin can be used to address the issue as suggested in [9, 13]. This approach shows a picture to the user according to the master password input. By checking if the picture is identical to the one from the last (correct) input, the user can verify the correctness of the master password. Unlike the ciphertext, the picture is not stored by the application, and thus will not be stolen from the online storage. Note the user does not need to remember the whole picture but just a vague impression. Other typo-correcting methods (e.g., [12]) can also be used in deployment without putting an extra burden on users.

2.4 Model-to-encoder transformation

Cheng et al. [14] propose a generic method to transform an arbitrary probability model to a *probability model transforming encoder*, which resists encoding attacks. Their core idea is to assume that messages are created by generating paths (i.e., a sequence of generating rules). Based on the idea, they formalize all current models for the single password or password vault. For example, in their formalization for PCFG models, the generating rules are production rules and the generating paths are leftmost derivations. To further encode a message, their encoder parses *all* generating paths of the message, randomly selects one path with its probability, and encodes each rule in the path. (In contrast, the existing encoders in [13, 18] use deterministic path selection, therefore it is easy to exclude the decoy seeds of which paths are not the deterministic ones.) In this way, each seed of this message can be randomly and uniformly picked. This feature is called *seed uniformity*, which is the "cure" to encoding attacks.

However, in some models associated with great ambiguity, a message may be generated by various paths. Parsing all these paths may yield heavy time complexity in encoding. Although Cheng et al. attempted to reduce the ambiguity by pruning some unnecessary paths, the low encoding performance still limits the scalability of their encoders.

3 Our Incrementally Updateable Scheme

We propose a generic construction for vault models and further construct an encoder, which provides incremental update for password vaults and achieves the update security (i.e., resisting intersection attacks). We summarize the main notions of this section in Table 2.

3.1 Our New Construction

In practice, the passwords in a vault $V = (pw_i)_{i=1}^{n}$ are generated one by one. Therefore, the probability $\Pr_{\text{real}}(V)$ can be expanded as

$$\Pr_{\text{real}}(V) = \prod_{i=0}^{n-1} \Pr_{\text{real}}(pw_{i+1} \mid pw_1, pw_2, \ldots, pw_i), \qquad (1)$$

where $\Pr_{\text{real}}(pw_{i+1} \mid (pw_{i'})_{i'=1}^{i})$ is the probability of creating a new password pw_{i+1} under the condition of given i

Table 2: Our proposed models and encoders

Probability model	Description
Password vault model $\Pr_{\text{PVM}}$	$\Pr_{\text{PVM}}(V)$ estimates the probability $\Pr_{\text{real}}(V)$ that a user generates V (as a real vault).
Multi-similar-password model $\Pr_{\text{MSPM}}$	$\Pr_{\text{MSPM}}(pw_{i+1} \mid pw_1, pw_2, \ldots, pw_i)$ estimates the probability $\Pr_{\text{real}}(pw_{i+1} \mid pw_1, pw_2, \ldots, pw_i)$ that a user generates a new password pw_{i+1} with i existing/old passwords $(pw_{i'})_{i'=1}^{i}$.
Single-similar-password model $\Pr_{\text{SSPM}}$	$\Pr_{\text{SSPM}}(pw' \mid pw)$ estimates the probability that a user generates a new password pw' by reusing/modifying an old password pw.
Single-password model $\Pr_{\text{SPM}}$	$\Pr_{\text{SPM}}(pw)$ estimates the probability that a user generates a brand new password pw (without reusing old passwords).
Unreused probability function f	$f(i)$ estimates the probability that the (user's) $i+1$-th password is not reused from the first i passwords.

Encoder	Description
For password vault model	$\mathsf{encode}(V)$ encodes the vault V.
For multi-similar-password model	$\mathsf{encode}(pw_{i+1} \mid pw_1, pw_2, \ldots, pw_i)$ encodes a new password pw_{i+1} given i existing/old passwords $(pw_{i'})_{i'=1}^{i}$.

old passwords $(pw_{i'})_{i'=1}^{i}$. Naturally, we can leverage a conditional probability model $\Pr_{\text{MSPM}}(\cdot|\cdot)$ to estimate the conditional probability and further construct a vault model. We denote $\Pr_{\text{MSPM}}(\cdot|\cdot)$ as *multi-similar password model*.

3.2 Conditional Probability Model Transforming Encoder

For a probability model with the following construction

$$\Pr_{\text{model}}((M_i)_{i=1}^{n}) = \prod_{i=1}^{n} \Pr_{\text{model}}(M_i \mid (M_{i'})_{i'=1}^{i-1}), \qquad (2)$$

using Cheng et al.'s model-to-encoder transformation can yield a probability model transforming encoder. However, the encoder has exponential time complexity if the model is ambiguous. Specifically, if there exit k_i paths to generate M_i from $(M_{i'})_{i'=1}^{i-1}$, then there will be $\prod_{i=1}^{n} k_i$ generating paths for $M = (M_i)_{i=1}^{n}$. Cheng et al.'s encoder has to calculate the probabilities of $\prod_{i=1}^{n} k_i$ paths and randomly select one with its probability, which yields the time complexity $O(\prod_{i=1}^{n} k_i)$.

To reduce the time complexity of the encoder, we extend Cheng et al.'s [14] transformation for conditional probability models. Since a conditional probability model can be seen as a probability model for each condition, a *conditional probability model transforming encoder (conditional encoder for short)* can be achieved by transforming the conditional probability model for each condition with Cheng et al.'s transformation.

By using the conditional encoder $(\mathsf{encode}(\cdot|\cdot), \mathsf{decode}(\cdot|\cdot))$ for $\Pr_{\text{model}}(\cdot|\cdot)$, we design a new encoder for $\Pr_{\text{model}}(\cdot)$ as follows:

1. To encode a message $M = (M_i)_{i=1}^{n}$: encode M_i to a seed S_i by $\mathsf{encode}(M_i \mid (M_{i'})_{i'=1}^{i-1})$, then output the concatenating of $\{S_i\}_{i=1}^{n}$, i.e., $S = S_1||S_2||\ldots||S_n$.
2. To decode a seed S: split S to $\{S_i\}_{i=1}^{n}$ according to the fixed length of S_i, decode M_i from S_i in order by $\mathsf{decode}(S_i \mid (M_{i'})_{i'=1}^{i-1})$, then output $M = (M_i)_{i=1}^{n}$.

In contrast to Cheng et al.'s encoder, our new encoder combined by conditional encoder only needs to select one path from k_i paths for $1 \leq i \leq n$, which significantly reduces the time complexity to $O(\sum_{i=1}^{n} k_i)$. In addition, since the conditional encoder is seed-uniform, our new encoder naturally inherits this property and therefore resists encoding attacks.

3.3 Incrementally Updateable Encoder

The proposed conditional encoder for the multi-similar-password model can encode a vault password by password[3]. This naturally brings an incremental update mechanism. In detail, we update the storage file of a vault as follows:

1. To add a new password: decrypt the vault, encode the new password by the conditional encoder for the multi-similar-password model, add the password seed to the tail of the vault seed, record the password position in plaintext, and finally encrypt the updated seed with the same (correct) key and the same nonce.
2. To delete an old password: mark the password as deleted (in plaintext) without changing the ciphertext.
3. To change an old password: delete the old password, add the new password as in Item 1, and update the password position for the corresponding account.

To achieve the update security, the PBE adopted in HE must satisfy the *prefix-keeping* property:

1. If a string str_1 is a prefix of a string str_2, then the ciphertext C_1 of str_1 is also a prefix of the ciphertext C_2 of str_2 with the same key and the same random nonce. Note the nonce is stored in plaintext, e.g., the salt for PBKDF and the initialization vector for CTR-mode.
2. If a ciphertext C_1 is a prefix of a ciphertext C_2 (with the same nonce), then the plaintext str_1 of C_1 is a prefix of the plaintext str_2 of C_2 under any (incorrect) key.

The PBE scheme used in [13, 14, 18], i.e., AES in CTR-mode merged with PBKDF, satisfies the prefix-keeping property. Thus, we use the same scheme for our design.

Intersection-attack resistance. Compared with the full update in the current honey vault schemes, our design decreases the time complexity in updating but also guarantees the update security against intersection attacks. As shown in Table 3, owing to the prefix-keeping property that our chosen PBE provides, we ensure that the old ciphertext is a prefix of its new version. Decrypting the old and new ciphertexts with the same (incorrect) master password, we will have that the new seed is the same as its old version except for the added tail (note if one leverages the schemes with full update, these two

[3] When initializing a vault with a set of existing passwords, the application can shuffle the passwords and then encode them one by one.

Table 3: The difference between old and new vaults after changing Facebook password[1]

	Position[2]	Ciphertext	Seed[3]	Passwords[3]
Previous	1	$C_1\|\ldots\|C_{10}$	$S_1\|\ldots\|S_{10}$	$pw_1,\ldots,pw_{10}$
Updated	11	$C_1\|\ldots\|C_{10}\|C_{11}$	$S_1\|\ldots\|S_{10}\|S_{11}$	$pw_1,\ldots,pw_{10},pw_{11}$

[1] We take the vault in Table 1 as an example and assume it contains 10 (human-generated) passwords.
[2] By position we refer to the password position of Facebook account.
[3] The previous and updated seeds/passwords are under a master password which may be the correct one or an incorrect one.

seeds will be totally different). Accordingly, the new decoy vault is identical to its old version except for the changed or added passwords. This means that the similarity of the old and new versions for each decoy vault is kept the same as that for the real vault. Therefore, if an attacker steals the old version of the vault storage file along with the current version, the attacker cannot get extra advantages and degenerates to an attacker only with the knowledge of the current version. This means the attacker cannot launch intersection attacks but only (traditional) distinguishing attacks based on the current version. We confirm this statement in an experiment with real-world datasets (see Appendix D).

Other potential threats. Unlike the existing schemes, our scheme maintains the password history (including the changed and deleted passwords), which may bring an advantage in distinguishing attacks. Based on users' password-changing habits, the attacker may leverage similarity among the old and new passwords for distinguishing. (Note the password-similarity attack proposed in Section 4 can be naturally extended for this purpose.) To address this issue, we can leverage a well-designed multi-similar-password model to capture the password-changing habits and further generate plausible-looking password histories for decoys. In addition, keeping the password history is provided by many real-world vault applications as a feature (rather than a flaw), e.g., LastPass [3]. Our scheme naturally provides this feature, while the existing ones cannot.

3.4 Multi-Similar-Password Model

To instantiate our construction for honey vault scheme, we need a *multi-similar-password model* $\Pr_{\mathrm{MSPM}}$ which can precisely estimate $\Pr_{\mathrm{real}}(pw_{i+1} \mid (pw_{i'})_{i'=1}^{i})$. To the best of our knowledge, such models do not exist in the literature. Pal et al. [38] mention this notion in their work on password guessing, but they leave it as future work without providing a specific model. Here, we give a simple design for the model.

Our design. We model a user's generation of a new password pw_{i+1} with i old passwords $(pw_{i'})_{i'=1}^{i}$ by a simplification that pw_{i+1} *either is created by "reusing" an old password (including a direct reuse or a slight modification of it) or is a brand new creation*. Accordingly,

$$\Pr_{\mathrm{MSPM}}(pw_{i+1} \mid pw_1, pw_2, \ldots, pw_i) \tag{3}$$
$$= f(i)\Pr_{\mathrm{SPM}}(pw_{i+1}) + \frac{1-f(i)}{i}\sum_{i'=1}^{i}\Pr_{\mathrm{SSPM}}(pw_{i+1} \mid pw_{i'}),$$

where $\Pr_{\mathrm{SSPM}}$, $\Pr_{\mathrm{SPM}}$ and $f(i)$ represent a *single-similar-password model*, a *single-password model*, and the *unreused probability function*, respectively. The single-password model captures the new generation, the single-similar-password model captures the reusing, and the unreused probability function is the probability that the user does not reuse i old ones.

Then we carefully instantiate the three components. For the single-password model, we choose to use a Markov model with well-set parameters (denoted as *Best-Markov*), since it performs the best under the single-password attack among existing single-password models [36, 37, 49, 52] (see Appendix E). For the single-similar-password model, we design a simple model which only captures the most prevalent password-reuse habit, i.e., head or tail modification [15]. For simplicity, this model regards two passwords as reused passwords if the length of their longest common substring is at least half of the maximum length of them (note we say the password pair has Feature LCSStr). In this way, our model is simple and further leads to the efficiency of encoding. The details of the model are given in Appendix A. We also try to use the existing single-similar-password models, e.g., the pass2path model [38] and the context Wasserstein autoencoder [39]. But we find our model is more suitable for honey vaults because of its best performance on the decoy vault generation (see Appendix G) and the high encoding efficiency (see Appendix A).

For the unreused probability function, we can leverage a real-world vault dataset (Pastebin, see Section 5.2) and count the empirical probability $\hat{f}(i)$ of the event that the $i+1$-th password is not reused from the first i passwords[4]. Further, we perform nonlinear regression on $\hat{f}(i)$ and find $\hat{f}(i)$ can be fitted well with a 3-degree rational function in the form

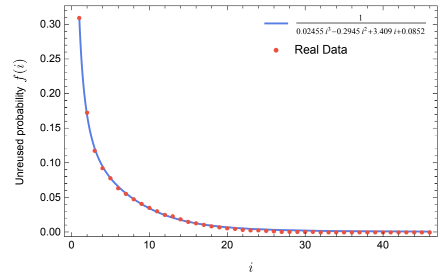


Figure 2: Unreused probability $f(i)$ that the $i+1$-th password is not reused from the first i passwords.

[4] There may be a great number of $i+1$-tuples for some i. Thus, we estimate the probability by sampling 10^4 tuples (with replacement) and counting $\hat{f}(i)$ in the samples instead of directly counting in all tuples.

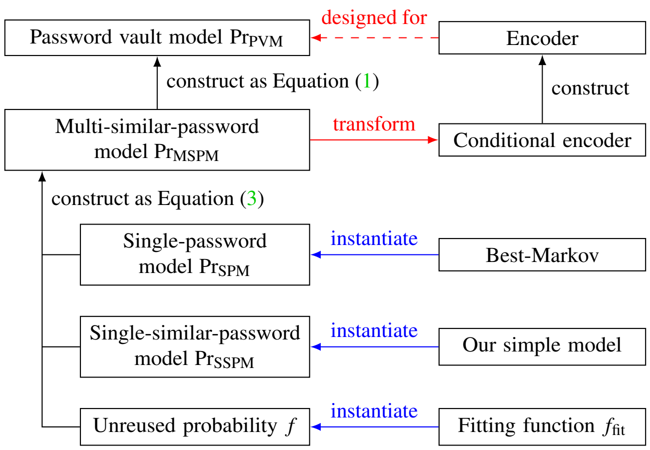


Figure 3: Technical roadmap of our designs.

$f(i) = 1/(\sum_{k=0}^{3} a_k i^k)$. As shown in Fig. 2, the fitted function $f_{\text{fit}}(i) = \frac{1}{0.02455i^3 - 0.2945i^2 + 3.409i + 0.0852}$ is very close to $\hat{f}(i)$, and $|f_{\text{fit}}(i) - \hat{f}(i)| \leq 4.101 \times 10^{-3}$. Thus, we use $f(i)$ as the unreused probability function (the coefficient is obtained from a training set, not Pastebin). In addition, $\hat{f}(i)$ decreases as i increases, indicating that the more passwords a user has, the lower probability the user creates a new password with. This reflects the limit of human memory on passwords.

With the above constructions and instantiations, we finally construct a concrete model and an encoder for honey vaults. The full construction is shown in Fig. 3.

Time complexity of encoding. Applying the transformation proposed in Section 3.2, we can get a conditional encoder for the multi-similar-password model and with efficiency in encoding. Specifically, pw_i is generated by reusing old passwords $(pw_{i'})_{i'=1}^{i-1}$ or is a brand new creation in our model. Therefore, there are at most i generating paths for pw_i under the condition of $(pw_{i'})_{i'=1}^{i-1}$. The time complexity of encoding a vault $(pw_i)_{i=1}^{n}$ is $O(n^2)$. The details of the conditional encoder are provided in Appendices A and B.

Complying with password policies. Many websites may adopt password policies to prevent users from using weak passwords, e.g., requiring passwords to contain at least 8 characters. The passwords generated by the vault models (i.e., in the decoy vaults) should always comply with the corresponding policies, otherwise, attackers will easily identify those decoys which do not.

For two classic policies, length restriction (e.g., ≥ 8 characters) and character requirement (e.g., the inclusion of an upper-case letter), we introduce an efficient method to adjust our model to guarantee that all passwords sampled from the model achieve the complying requirement. The core idea is to exclude the noncompliant lengths or characters when encoding and decoding. Specifically, we adjust the length distribution in best-Markov and our single-similar-password model; and meanwhile, we model the position of the required character type and adjust the corresponding character distribution (by adjusting, we mean excluding the lengths or characters not complying the requirement and re-normalizing the probabilities of the rest ones).

We note it is very challenging to guarantee the need w.r.t. more complex policies (e.g., blacklist or password strength requirement) and we leave this as an open problem. For the websites with these policies, users can use randomly-generated passwords that are easily complied with the policies.

3.5 Leakage Detection

We propose a mechanism to detect the leakages of storage files of honey vaults. The core idea is to generate and store some decoy accounts (called *honeypot accounts*) in a user's real vault. For example, if the user has a real account(name) "Alice07" on Google, the vault application may generate (and register) a honeypot account "Alice07" on Yahoo (with a password generated by our model). These honeypot accounts will not be used by the user (note we can choose the websites the user rarely visits for honeypot accounts to avoid the user's misuse); and meanwhile they are really registered on the corresponding websites. An attacker with the stolen storage file cannot tell them from real accounts and will probably log in to them (for online verification). Once the logins of honeypot accounts occur, the leakage can be reliably detected. Then the user should change all passwords in the vault to prevent consequent account compromise. In this way, we significantly mitigate the risk of vault file leakages.

Further, we consider the threat in the case where a password in a vault is leaked as well as the vault storage file. In this case, the attacker can offline tell the real vault by checking if the leaked password is in the vault. Although this is an important and practical threat, it is not considered in [13, 14, 18]. Fortunately, leveraging the leakage detection mechanism for honey vaults and the existing alert mechanisms for password breaches (e.g., [1, 19]), we can detect the password and vault leakages, respectively. This enables users to timely change the leaked passwords or vaults. Our solution is a "pre-action" mechanism for the threat (preventing it from happening) rather than a post-action (resisting the attacks after the threat is there). Details are given in Appendix C.

3.6 Implementation and Performance

We use Python 3.8.2 with Cryptography 2.9 to implement our vault scheme. Since AES within CTR mode satisfies our requirement, we adopt it for encryption. For key derivation, we use PBKDF2 with SHA-256. To evaluate the efficiency of our scheme, we run it on a laptop, MacBook Pro with 2.6GHz

Table 4: Our proposed attacks

Scheme	Attack	Description	Priority function
Static (and adaptive)	Optimal strategy	Exploiting the difference between the real and decoy vaults.	Equation (4). One factor in it is the real-to-decoy probability ratio on the vault (the ratio of the real probability to the decoy probability of the vault).
	Single-password attack	Exploiting the difference on the single-password distributions.	$p_{\mathrm{SP}}(V_i)$, i.e. Equation (9). It is the product of real-to-decoy probability ratios for each password in the vault V_i with smoothing.
	Password-similarity attack	Exploiting the difference on the password-similarity features.	$p_{\mathrm{PS}}(V_i)$, i.e. Equation (10) with $\mathcal{F} = \{\mathrm{M}\backslash\mathrm{I}, \mathrm{I}\backslash\mathrm{M}\}$. It is the product of real-to-decoy probability ratios for each password-similarity feature in $\mathcal{F}$.
	Hybrid attack	Combined by single-password attack and password-similarity attack.	$p_{\mathrm{H}}(V_i)$, i.e. Equation (11). It is the product of the priority functions of the single-password attack and the password-similarity attack.
Adaptive	Optimal strategy	Exploiting the model/encoder which is adjusted according to the real vault.	Equation (13). It contains an extra factor—the probability of the adjusted model/encoder if the vault is real.
	Adaptive extra attack	Exploiting the number of n-grams of which probabilities are increased.	$p_{\mathrm{AE}}(V_i)$, i.e. Equation (14). It is the real-to-decoy probability ratio on the increased n-gram number (under the condition V_i is real or not).
	Adaptive hybrid attack	Combined by hybrid attack and adaptive extra attack.	$p_{\mathrm{AH}}(V_i)$, i.e. Equation (15). It is the product of the priority functions of the hybrid attack and the adaptive extra attack (with a little modification).

Table 5: The performance of our vault scheme

Vault size	2	20	200	2,000
Encode the last password[1]	1.21 ms	1.57 ms	7.88 ms	65.81 ms
Decode the last password[1]	0.28 ms	0.04 ms	0.03 ms	0.05 ms
Encode the vault	2.12 ms	26.27 ms	978.46 ms	70,782.77 ms
Decode the vault	1.13 ms	2.72 ms	13.21 ms	264.83 ms
Encrypt the vault[2]	2.27 ms	27.90 ms	951.44 ms	71,252.72 ms
Recover the vault[2]	1.53 ms	3.40 ms	13.51 ms	329.89 ms
Add a password[3]	1.05 ms	1.76 ms	7.97 ms	78.96 ms
Recover a password[4]	1.12 ms	1.28 ms	1.47 ms	3.20 ms
Storage file	0.33 KB	3.30 KB	33.00 KB	330.00 KB

[1] Using our conditional encoder with all previous passwords.
[2] Including encoding/decoding operations.
[3] Including encoding and encrypting operations. Note due to the prefix-keeping property of the encryption scheme, we only need to encrypt the seed of the new passwords instead of the seed of the whole vault.
[4] From the ciphertext of the whole vault and including decoding and decrypting operations. Note using our encoder, we do not need to decode the whole vault. Instead, to decode a password *pw*, we only need to decode the previous passwords reused for *pw*.

Intel Core i7 and 16 GB memory. The average times of 1,000 runnings are presented in Table 5.

Our incrementally updateable encoder can encode a new password in 65.81 ms even with 2,000 old passwords, which is extremely efficient. With our encoder, a new password can be efficiently added to the vault (78.96 ms for size 2,000). Although encoding a vault of size 2,000 requires 70.78 s, it is still practical in stand-alone use. This is because this operation is needed only when a vault is initialized. After initialization, the user only needs to incrementally encode the new or changed passwords.

In our scheme, decoding and recovering passwords are more efficient than encoding and encrypting them. Recovering a password in a vault of size 2,000 only needs 3.20 ms. This is efficient because we made optimization for this operation. A naive way to decode the i-th password pw_i in a vault is to decode the first $i-1$ passwords one by one, which brings $O(i)$ time complexity. In contrast, we decode the first rule of pw_i and know that pw_i is reused from a previous password pw_j or none of the previous ones. Then we can decode pw_i via decoding pw_j or directly. Our decoding method reduces the time complexity to $O(\log(i))$.

4 Attacks Against Honey Vault Schemes

To evaluate the security of honey vault schemes when the storage file (only the current version) is stolen, we investigate the theoretically optimal strategy for online verifications, and further propose several practical attacks. We summarize them in Table 4 to make them more clear.

4.1 Attacker Model

Attacker ability. We consider a significant threat for honey vaults: an attacker steals the (current) storage file of a vault (e.g., from online sync services), and tries to reveal all stored passwords from the file. The attacker also gets the program of the honey vault scheme, including the HE algorithm and the encoder, since the program should be stored along with the storage file to provide handy service to users. So the attacker can try to decrypt the ciphertext with a dictionary of master password guesses. To distinguish the real and decoy vaults decrypted from the ciphertext, the attacker may leverage some public information, e.g., leaked datasets, website password restrictions. Note that if the vault scheme uses a public dataset to train its vault model, the attacker can identify the dataset from the encoder of the model and may further use it to launch attacks.

In this section, we do not consider the cases where the attacker additionally gets an old version of the storage file or a website password in the vault. We have addressed these two cases in Sections 3.3 and 3.5, respectively.

Attack process. As shown in Algorithm 1, to reveal passwords from an encrypted vault, the attacker should decrypt the ciphertext c with a dictionary of the master password guesses $\{mpw_i\}_{i=1}^N$, and then obtain a (large) group of vaults $\{V_i\}_{i=1}^N$,

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.
Output: the real password vault of c.

```
Offline:
    for i ← 1 to N do
        Decrypt c with mpw_i and get a vault V_i.
    Sort {V_i}_{i=1}^N in descending order of p(V_i) and get a list {V'_i}_{i=1}^N.
Online:
    for i ← 1 to N do
        Online verify the correctness of V'_i (i.e., log in with a password
          in V'_i on the corresponding website).
        If V'_i is correct, output V'_i.
```

in which at most one of the vaults is real. To check the correctness of the vaults, the attacker needs to log in with the passwords in the vaults (i.e., online verification).

The effectiveness of the attack depends on 1) the offline guessing order of master passwords and 2) the online verification order of the vaults. The former is mainly related to the strength of the master password, while the latter relies on the security of the encoder (i.e. the indistinguishability of real and decoy vaults). We recall that a master password is a human-memorable password and may suffer from general password guessing attacks [36,38,51,52]. We take the same research direction as [13, 14, 18], focusing on the security of encoders.

We assume attackers will test target vaults (via online verification) in a descending order defined by a priority function. Each vault is assigned a priority value so that attackers first online verify those vaults with *greater* values. We denote this priority function as p with a subscript representing the name abbreviation of the attack. In practice, $p(V_i)$ is strongly related to the probability that V_i is the real vault among $\{V_i\}_{i=1}^{N}$.

4.2 Theoretically Optimal Strategy

The theoretically optimal strategy of online verification is to verify the vaults in the descending order of conditional probabilities, where the condition is all the information that attackers have known. In the following, we investigate this strategy by analyzing the conditional probabilities.

We denote the random variables of the (real) master password, the (real) vault, the (real) seed and the ciphertext, as MPW, V, S and C, respectively. Let $MPW = mpw_i$ denote the event that a user's real master password is identical to mpw_i (i.e., the user chooses mpw_i as the master password MPW). Similarly, we define $V = V_i$, $S = S_i$ and $C = c$. To keep consistency with notations in Section 4.1, we define mpw_i, V_i, S_i, c such that S_i is decrypted from c with 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), and the probability of encoding V_i to S_i as $\Pr_{\text{encode}}(S_i \mid V_i)$ (i.e., $\Pr(S = S_i \mid V = V_i)$), respectively.

For the (static) honey vault schemes, the only information attackers can learn about the real vault V is its ciphertext c. (An adaptive scheme will leak extra information about V, see Section 4.4.) Therefore, *the optimal strategy is to verify V_i in the descending order of* $\Pr(V = V_i \mid C = c)$.

For simplicity, we use $\Pr(MPW = mpw_i \mid C = c)$ to estimate $\Pr(V = V_i \mid C = c)$[5]. In addition, we notice that the existing honey vault schemes [13, 14, 18] require a user to create a completely new and distinct master password so that the master password is independent of all the passwords in the vault. This requirement is due to the limitation of HE [23] that cannot guarantee the security for dependent key and message distributions. Therefore, $\Pr(MPW = mpw_i, V = V_i) = \Pr_{\text{MPW}}(mpw_i)\Pr_{\text{real}}(V_i)$. According to the Bayesian theorem, we have the following theorems.

Theorem 1. *For an arbitrary encoder,*

$$\begin{aligned}&\Pr(MPW = mpw_i \mid C = c)\\ &\quad = k \cdot \Pr_{\text{MPW}}(mpw_i)\Pr_{\text{real}}(V_i)\Pr_{\text{encode}}(S_i \mid V_i),\end{aligned} \tag{4}$$

where k is a constant independent of i.

Proof. For arbitrary encoder we have

$$\begin{aligned}&\Pr(MPW = mpw_i \mid C = c)\\
=&\frac{\Pr(C = c, MPW = mpw_i)}{\Pr(C = c)}\\
=&\frac{\Pr(C = c, MPW = mpw_i, V = V_i)}{\Pr(C = c)}\\
=&\frac{\Pr(C = c \mid MPW = mpw_i, V = V_i)}{\Pr(C = c)}\\
&\times \Pr(MPW = mpw_i, V = V_i)\\
=&\frac{\Pr(S = S_i \mid V = V_i)\Pr(C = c \mid S = S_i, MPW = mpw_i)}{\Pr(C = c)}\\
&\times \Pr_{\text{MPW}}(mpw_i)\Pr_{\text{real}}(V_i)\\
=&\frac{\Pr(C = c \mid S = S_i, MPW = mpw_i)}{\Pr(C = c)}\\
&\times \Pr_{\text{MPW}}(mpw_i)\Pr_{\text{real}}(V_i)\Pr_{\text{encode}}(S_i \mid V_i).\end{aligned}$$

$\Pr(C = c \mid S = S_i, MPW = mpw_i)$ only depends on the traditional PBE used by the honey vault scheme and is a constant independent of i. Take the PBE used by Chatterjee et al. [13] and Golla et al. [18] as an example, i.e., AES in CTR-mode with PBKDF,

$$\Pr(C = c \mid S = S_i, MPW = mpw_i) = \frac{1}{s_{\text{IV}} s_{\text{salt}}},$$

[5] The vaults V_i and V_j under different master passwords mpw_i and mpw_j may be the same (i.e., $V_i = V_j$). But this only happens with a low probability, especially if the vaults are of a large size, because the space of vaults is much larger than that of master passwords.

where s_{IV}, s_{salt} are the sizes of the initialization vector space and the salt space, respectively. Therefore,

$$k = \frac{\Pr(C = c \mid S = S_i, MPW = mpw_i)}{\Pr(C = c)}$$

is a constant independent of i. $\square$

This theorem shows that Cheng et al.'s strong encoding attack [14] is a degenerate case of the optimal strategy by only considering the last factor $\Pr_{\mathsf{encode}}(S_i \mid V_i)$ as the priority function. If an encoder is not seed-uniform (i.e., the seeds of a message are not randomly chosen when encoding the message), the real and decoy vaults can be effectively distinguished by merely exploiting the encoder (i.e., calculating $\Pr_{\mathsf{encode}}(S_i \mid V_i)$) without any knowledge of the master password and password vault distributions (i.e., $\Pr_{\mathrm{MPW}}(mpw_i)$, $\Pr_{\mathrm{real}}(V_i)$).

Theorem 2. *If the encoder is seed-uniform, then*

$$\begin{aligned}\Pr(&MPW = mpw_i \mid C = c) \\ &= k \cdot \Pr_{\mathrm{MPW}}(mpw_i) \frac{\Pr_{\mathrm{real}}(V_i)}{\Pr_{\mathrm{decoy}}(V_i)},\end{aligned} \tag{5}$$

where k is a constant independent of i.

Proof. Let l be the length of seeds. Due to the seed-uniformity of the encoder, $\Pr_{\mathsf{encode}}(S_i \mid V_i) = \frac{1}{2^l \Pr_{\mathrm{decoy}}(V_i)}$ ($2^l \Pr_{\mathrm{decoy}}(V_i)$ is the number of seeds for V_i). Then we have

$$\begin{aligned}&\Pr(MPW = mpw_i \mid C = c) \\ =&\frac{\Pr(C = c \mid S = S_i, MPW = mpw_i)}{2^l \Pr(C = c)} \\ &\times \Pr_{\mathrm{MPW}}(mpw_i) \frac{\Pr_{\mathrm{real}}(V_i)}{\Pr_{\mathrm{decoy}}(V_i)}.\end{aligned}$$

We can let k be

$$\frac{\Pr(C = c \mid S = S_i, MPW = mpw_i)}{2^l \Pr(C = c)},$$

which is a constant independent of i. $\square$

Theorem 2 indicates that if an encoder is seed-uniform, attackers cannot get any information from the encoder except the decoy vault distribution (i.e., $\Pr_{\mathrm{decoy}}(V_i)$). This analysis confirms that Cheng et al.'s transformation is secure, i.e., their encoder resists encoding attacks. By applying the secure encoders to the existing honey vault schemes [13, 18], we only need to consider how to hold against *distribution difference attacks*. This type of attack is defined in [14], referring to the attacks that only exploit the difference between the real and decoy distributions (i.e., $\Pr_{\mathrm{real}}(V_i)$ and $\Pr_{\mathrm{decoy}}(V_i)$).

We use Theorem 2 to present a new vision for the security analysis w.r.t. HE and honey vault schemes. For a seed-uniform encoder, if the decoy distribution is the same as the real one, i.e., $\Pr_{\mathrm{decoy}} = \Pr_{\mathrm{real}}$, then $\Pr(MPW = mpw_i \mid C = c) = \Pr_{\mathrm{MPW}}(mpw_i)$ (in this case, $k = 1$). Therefore, the mutual information of C and MPW is $I(MPW; C) = H(MPW) - H(MPW \mid C) = 0$. This means that the ciphertext C does not leak any information of the key MPW, which achieves the ideal security of HE and honey vault schemes.

Table 6: Examples of real-to-decoy probability ratios

	Vault	Password	Reuse feature[†]		Increased n-gram number
Example	(123456,1234567)	123456	0	1	4
Real probability	–[*]	10^{-2}	0.8	0.2	10^{-7}
Decoy probability	10^{-4}	5×10^{-3}	0.4	0.6	10^{-9}
Ratio	–[*]	2	2	0.333	100

[*] It is hard to precisely calculate the real probability and the ratio for a vault. So we use some methods to estimate the ratio for attacks.
[†] Each feature used in our classifier is a Boolean (binary variable).

Without considering $\Pr_{\mathrm{MPW}}$ (as discussed in Section 4.1), *the optimal online verification order for vaults $\{V_i\}_i$ is the descending order of*

$$\frac{\Pr_{\mathrm{real}}(V_i)}{\Pr_{\mathrm{decoy}}(V_i)}, \tag{6}$$

which is denoted as p_{OPT}. The basic idea behind this *real-to-decoy probability ratio* p_{OPT} is simple and intuitive. A high $p_{\mathrm{OPT}}(V_i)$ means the vault model used in the honey vault scheme significantly underestimates the real probability of V_i. In other words, V_i is less likely generated by the vault model than being generated by the user. Therefore, V_i is more likely to be real among $\{V_i\}_{i=1}^N$.

4.3 Practical Attacks

The optimal online verification with the priority function p_{OPT} is hard to be carried out, since it is difficult to precisely calculate the real probability $\Pr_{\mathrm{real}}(V_i)$ for attackers. This difficulty also hinders the direct use of existing techniques, e.g., Bayesian updating, in calculating p_{OPT}.

Leveraging an advanced model seems to be a straightforward method to estimate $\Pr_{\mathrm{real}}$. Unfortunately, all current models have defects, which lead to the misestimation of the probabilities [50]. For example, the PCFG model [52] underestimates the passwords with relative components, e.g., "1q2w3e", because the model assumes these components are independent and does not further consider their relationships [50]. The misestimation of a model will lead to the misestimation of $p_{\mathrm{OPT}}(V)$ and further significantly decrease the effectiveness of attacks. Note we have tried to use different single-password models to estimate the real single-password distribution, e.g., Markov models [36], but could not obtain stable and reasonable effectiveness in attacking all other single-password models.

To overcome the difficulty, we use several methods to estimate the real-to-decoy probability ratio p_{OPT} and further propose several practical attacks as follows. The characterization of both the single-password distribution and the password similarity are the two significant indices of a vault model. Accordingly, our estimations will focus on these two indices.

Single-password attack. To capture the difference between real and decoy vaults on the single-password distribution, we use the real-to-decoy probability ratio on the single password to estimate the ratio $p_{\text{OPT}}(V_i)$ on vault. Formally assume that the passwords in a vault are independent (ignoring their similarity), $p_{\text{OPT}}(V_i)$ can be simplified/estimated as

$$\prod_{pw \in V_i} \frac{\Pr_{\text{real}}(pw)}{\Pr_{\text{decoy}}(pw)}, \tag{7}$$

where $\Pr_{\text{real}}(pw)$ and $\Pr_{\text{decoy}}(pw)$ represent the real and decoy single-password distributions, respectively. $\Pr_{\text{decoy}}(pw)$ can be calculated by the single-password models in honey vault schemes, but we still need to estimate $\Pr_{\text{real}}(pw)$.

To estimate $\Pr_{\text{real}}(pw)$, we directly use the relative frequency of pw in a password training set, instead of using some password models. We note this is because we do not want to bring the misestimation of password probability yielded by single-password models to our attacks (as discussed above). According to the law of large numbers, the relative frequency of an event converges (almost surely) to its probability, as the number of experiments approaches infinity. To avoid misestimation incurred by inappropriate training sets, we choose the dataset which has been used to train the single-password model by the honey vault schemes (see Section 5.2).

In addition, smoothing is further needed since some passwords not appearing in the training set have zero frequency. With a carefully-designed smoothing method, we propose an estimation $p_{\text{SP}}(pw)$ for $\frac{\Pr_{\text{real}}(pw)}{\Pr_{\text{decoy}}(pw)}$ as

$$p_{\text{SP}}(pw) = \begin{cases} 1 & \text{if } f_{\text{a}}(pw) \le f_{\text{d}} \text{ and } \frac{f'_{\text{r}}(pw)}{\Pr_{\text{decoy}}(pw)} > 1, \\ \dfrac{f'_{\text{r}}(pw)}{\Pr_{\text{decoy}}(pw)} & \text{otherwise,} \end{cases} \tag{8}$$

where $f_{\text{a}}(pw)$ is the absolute frequency of pw in the training set, n is the size of the training set, α_{s} is a smoothing parameter, f_{d} is a parameter representing the demarcation line between high-frequency and low-frequency passwords, and $f'_{\text{r}}(pw) = \frac{f_{\text{a}}(pw)+\alpha_{\text{s}}}{n+\alpha_{\text{s}}}$. Our estimation is similar to maximum likelihood estimation (MLE) with Laplace smoothing. Unlike Laplace smoothing used in [36], our smoothing only adds α_{s} for the calculated pw instead of all passwords in the password space. This is because the password space is extremely large, using Laplace smoothing will make $f'_{\text{r}}(pw)$ to approach to 0 for all pw. In addition, we note that our smoothing may lead to overestimation for the probabilities of some low-frequency passwords. Thus, we choose to set $p_{\text{SP}}(pw) = 1$ for passwords with absolute frequency no more than f_{d} and $\frac{f'_{\text{r}}(pw)}{\Pr_{\text{decoy}}(pw)} > 1$. After a few tries, we finally set $\alpha_{\text{s}} = 1$ and $f_{\text{d}} = 5$.

Using $p_{\text{SP}}(pw)$, we propose a *single-password attack* with the following priority function

$$p_{\text{SP}}(V_i) = \prod_{pw \in V_i} p_{\text{SP}}(pw). \tag{9}$$

Informally, this attack gives priority to the vaults of which some passwords are not accurately characterized (their probabilities are underestimated) by the single-password model in honey vault schemes. According to Theorem 2 and Equation (7), the vaults are more likely to be real.

Password-similarity attack. To capture the difference between real and decoy vaults on the password similarity, we use the real-to-decoy probability ratio on some features with a Bernoulli naive Bayes classifier to estimate the ratio $p_{\text{OPT}}(V_i)$. The features should capture the misestimation of the vault models on the password similarity. With a carefully-chosen feature set $\mathcal{F}$, $p_{\text{OPT}}(V_i)$ can be simplified/estimated as

$$\prod_{F \in \mathcal{F}} \frac{\Pr_{\text{real}}(F = F(V_i))}{\Pr_{\text{decoy}}(F = F(V_i))}, \tag{10}$$

where $F(V_i)$ is the value of Feature F for V_i ($F(V_i) = 1$ if V_i has Feature F, otherwise, $F(V_i) = 0$), $\Pr_{\text{real}}(F = x)$ is the probability that the value of Feature F is x for a real vault, and $\Pr_{\text{decoy}}(F = x)$ is the probability for a decoy vault.

We demonstrate that the estimation is effective. Unlike $\Pr_{\text{real}}(V_i)$ which is difficult to be calculated, $\Pr_{\text{real}}(F = x)$ can be counted from a password vault dataset (counting the proportion of vaults which have Feature F for $x = 1$, and the rest proportion for $x = 0$). The vault dataset (Pastebin, in Section 5.2) we are going to use is small, so we only choose two binary features for the Bayes classifier with four parameters. Similarly, $\Pr_{\text{decoy}}(F = x)$ can be counted from a set of decoy vaults generated by the encoder (decoding random seeds).

To design appropriate features, we first analyze the characterization of vault models on the password similarity. Recall that a user almost always reuses passwords in different accounts [15], therefore, the passwords in his vault usually are similar. A vault model should precisely capture the similarity and further generate similar (i.e., reused) passwords in decoy vaults. In the existing models, a password pair (pw_1, pw_2) is treated as similar by simple rules: in Golla et al.'s model [18], pw_1, pw_2 are treated as similar if pw_1 is the same as pw_2 except for the last 5 characters (we say (pw_1, pw_2) has Feature GM); in Chatterjee et al.'s model [13], pw_1, pw_2 are treated as similar if pw_1 and pw_2 share at least one production rule in their PCFG model (we say (pw_1, pw_2) has Feature CM). The sample treatment leads to the inaccuracy of the models on password similarity.

To crack a vault model, we define Features M and I: (pw_1, pw_2) has Feature M if the model treats pw_1, pw_2 as similar; (pw_1, pw_2) has Feature I if a user can create pw_2 by reusing pw_1. Then Features M and I capture the similarity among passwords in decoy vaults and real vaults, respectively. Therefore, the difference between Features M and I can be used to exploit the misestimation of the model. Formally, we define the feature difference as follows.

Definition 1. *We say (pw_1, pw_2) has Feature A\B, i.e., the difference of Features A and B, if (pw_1, pw_2) has Feature A but not Feature B.*

With a well-defined Feature I, we can use $\mathcal{F} = \{\text{M\textbackslash I}, \text{I\textbackslash M}\}$ to propose a password-similarity attack. Here, we define that a vault V has Feature X, if there exist two passwords pw_1, pw_2 in V such that (pw_1, pw_2) has Feature X. Note that the model probably overestimates the probability of a vault with Feature M\I and underestimates that for a vault with Feature I\M.

However, it is difficult to precisely define Feature I. We use Feature LCSStr as an approximation of Feature I to attack Chatterjee et al.'s and Golla et al.'s schemes, due to the fact that modifying the head or tail characters is most popular in reuse habits [15]. But for our scheme, Feature M is Feature LCSStr (see Section 3.4), then Features M\I and I\M are trivial with Feature LCSStr as Feature I (no vaults have Feature LCSStr\LCSStr). So we leverage four password similarity meters used in [15] to define Feature I, including Levenshtein [31], longest common subsequence (LCS), Manhattan [28], and Overlap [30]. For each meter F, we define that (pw_1, pw_2) has Feature F, if the similarity score of (pw_1, pw_2) is at least 0.5 under the meter F.

To summarize, we use Equation (10) as the priority function p_{PS} with $\mathcal{F} = \{\text{M\textbackslash I}, \text{I\textbackslash M}\}$ for the *password-similarity attack*. To crack Chatterjee et al.'s scheme, Features M and I are Features CM and LCSStr, respectively; for Golla et al.'s scheme, Features M and I are Features GM and LCSStr, respectively; for our scheme, Features M is Feature LCSStr and Feature I is one of Features Levenshtein, LCS, Manhattan, and Overlap. We summarize these features in Table 7. Note that this attack gives priority to these vaults, in which the password similarity is not well characterized by the vault model in a honey vault scheme. According to Theorem 2 and Equation (10), these vaults are more likely to be real.

Hybrid attack. Combining the single-password attack with the password-similarity attack, we propose a hybrid attack with the following priority function

$$p_{\text{H}}(V_i) = p_{\text{SP}}(V_i) \cdot p_{\text{PS}}(V_i). \tag{11}$$

Note that like p_{OPT}, p_{SP} and p_{PS} are in the form of real-to-decoy probability ratios, but on different indices. So we keep this form by multiplying p_{SP} and p_{PS}, and then the product p_{H} can estimate p_{OPT} more precisely. This is confirmed by our experimental results that the hybrid attack always performs better than the previous two attacks (see Section 5).

Table 7: The features of a password pair used in the password-similarity attack

Feature	Description
LCSStr	The password pair has Feature LCSStr if the length of their longest common substring is at least half of the maximum length of them.
GM	If the two passwords are the same except for the last 5 characters.
CM	If the two passwords share at least one production rule in Chatterjee et al.'s PCFG model.
Levenshtein	If the Levenshtein (edit) distance of the two passwords is at least half of the maximum length of them.
LCS	If the length of the longest common subsequence of the two passwords is at least half of the maximum length of them.
Manhattan	The Manhattan distance of the two passwords is at least half of the length sum of them.
Overlap	The union size of the character sets of the two passwords is at least half of the minimal size of the character sets of them.

Other attacks. The support vector machine (SVM) attack and the Kullback–Leibler (KL) divergence attack are proposed by Chatterjee et al. [13] and Golla et al. [18], respectively. Since the latter outperforms the former against all the existing vault models, we will use the latter for comparison. The priority function of the KL divergence attack is defined as

$$p_{\text{KL}}(V_i) = \sum_{j=1}^{s} f_j \log \frac{f_j}{\Pr_{\text{decoy}}(pw_j)}, \tag{12}$$

where V_i contains s unique passwords $\{pw_j\}_{j=1}^{s}$, and f_j is the relative frequency of pw_j in V_i.

Golla et al. [18] exploit extra information to enhance their KL divergence attack, including username, password reuse rate and password policy. Among the three types of information, only password policy has significant improvement for KL divergence attack. Attackers can easily exploit it to distinguish the decoys not complying with the policy. We will also consider the *password policy attack* with a minor difference. Formally speaking, the priority function p_{PP} of this attack can be defined as: 1) if there exists a password not complying with its policy in the vault V_i, then $p_{\text{PP}}(V_i) = 0$; 2) otherwise, $p_{\text{PP}}(V_i) = 1$. Unlike [18] which only considers one password in a vault, our password policy attack requires all passwords to comply with their policies and can exclude much more decoy vaults.

4.4 More Attacks to Adaptive Encoders

For static schemes, the storage file c is the only information that attackers can learn. But for adaptive schemes, attackers can learn extra information about the real vault from the encoder. This is because the adaptive encoder is adjusted according to the encrypted real vault. We exploit the "extra" information and propose more attacks to adaptive schemes.

Theoretically optimal strategy. We denote the random adaptive encoder as DTE, and let $DTE = DTE^*$ be the event that the encoder is adjusted to DTE^* according to the real vault. *The theoretically optimal strategy here is to verify the vaults* $\{V_i\}_i$ *in the descending order of* $\Pr(V = V_i \mid DTE = DTE^*, C = c)$. Recall that the adaptive encoder DTE^* and the ciphertext c are the only information that attackers can learn. Similar to the attacks against static encoders, we leverage $\Pr(MPW = mpw_i \mid DTE = DTE^*, C = c)$ to estimate $\Pr(V = V_i \mid DTE = DTE^*, C = c)$ and have the following theorem.

Theorem 3. *If the adaptive encoder DTE^* is seed-uniform, then*

$$\begin{aligned}&\Pr(MPW = mpw_i \mid DTE = DTE^*, C = c)\\ =&k \cdot \Pr_{\mathrm{DTE}}(DTE^* \mid V_i) \Pr_{\mathrm{MPW}}(mpw_i) \frac{\Pr_{\mathrm{real}}(V_i)}{\Pr_{DTE^*}(V_i)},\end{aligned} \tag{13}$$

where k is a constant independent of i, and $\Pr_{DTE^*}(V_i)$ represents the distribution of decoy vaults generated by DTE^*, $\Pr_{\mathrm{DTE}}(DTE^* \mid V_i)$ represents the probability that DTE *is adjusted to DTE^* according to V_i.*

Proof. Let $\Pr_{\mathrm{DTE}}(DTE^* \mid V_i)$ denote $\Pr(DTE = DTE^* \mid V = V_i)$, i.e., the conditional probability that the DTE is adjusted to DTE^* according to the encrypted real vault V_i. Then we have

$$\begin{aligned}&\Pr(MPW = mpw_i \mid DTE = DTE^*, C = c)\\
=&\frac{\Pr(DTE = DTE^*, C = c, MPW = mpw_i)}{\Pr(DTE = DTE^*, C = c)}\\
=&\frac{\Pr(DTE = DTE^*, C = c, MPW = mpw_i, V = V_i)}{\Pr(DTE = DTE^*, C = c)}\\
=&\frac{\Pr(C = c \mid DTE = DTE^*, MPW = mpw_i, V = V_i)}{\Pr(DTE = DTE^*, C = c)}\\
&\times \Pr(DTE = DTE^*, MPW = mpw_i, V = V_i)\\
=&\frac{\Pr(C = c \mid S = S_i, MPW = mpw_i)}{\Pr(DTE = DTE^*, C = c)}\\
&\times \Pr(S = S_i \mid DTE = DTE^*, V = V_i)\\
&\times \Pr_{\mathrm{DTE}}(DTE^* \mid V_i) \Pr_{\mathrm{MPW}}(mpw_i) \Pr_{\mathrm{real}}(V_i)\\
=&\frac{\Pr(C = c \mid S = S_i, MPW = mpw_i)}{2^l \Pr(DTE = DTE^*, C = c)}\\
&\times \Pr_{\mathrm{DTE}}(DTE^* \mid V_i) \Pr_{\mathrm{MPW}}(mpw_i) \frac{\Pr_{\mathrm{real}}(V_i)}{\Pr_{DTE^*}(V_i)}.\end{aligned}$$

We can set k to be

$$\frac{\Pr(C = c \mid S = S_i, MPW = mpw_i)}{2^l \Pr(DTE = DTE^*, C = c)},$$

which is a constant independent of i. □

The priority functions of optimal strategy for the static and adaptive encoders differ by one factor $\Pr_{\mathrm{DTE}}(DTE^* \mid V_i)$, which indicates the "extra" information leaked by DTE^*.

Practical attacks. To carry out practical attacks, we need to calculate $\Pr_{\mathrm{DTE}}(DTE^* \mid V_i)$ individually. But this is difficult for Golla et al.'s adaptive encoder [18]. We propose to use a simple method to estimate its value, and further design two attacks. The estimation method is similar to the one used in [18] for the security analysis of adaptive encoders. The basic idea of the estimation is to leverage the real-to-decoy probability ratio on the number of n-grams whose probability is increased by Golla et al.'s adjustment.

We first review Golla et al.'s adjusting process: 1) for each password in a real vault, randomly pick an n-gram (from the password) and multiply its probability with a factor α; 2) for each un-increased n-gram, increase its probability by the factor α with a probability of p_{i}. Golla et al. empirically set $n = 4$ for the Golla-Markov, and $\alpha = 5$ and $p_{\mathrm{i}} = 0.2$. Note that there exists an ambiguity that if the probability of an n-gram can be increased multiple times, when the n-gram appears in multiple passwords. If we assume that is a "yes", then an n-gram increased t times ($t \geq 2$) must appear in at least t passwords in the real vault. Noticing this, attackers can directly identify the real vault with this type of n-gram.

For real password vaults containing multiple identical or similar passwords, multiple increases will occur with a high probability. If a vault contains k identical passwords of length l, then the multiple increases happen with at least $1 - \frac{l^{\underline{k}}}{l^k}$ probability, where $l^{\underline{k}}$ is the falling factorial representing $\prod_{i=0}^{k-1}(l-i)$. Note that if $k > l$, then $1 - \frac{l^{\underline{k}}}{l^k} = 1$. Therefore, allowing multiple increases may be a great threat for Golla et al.'s adaptive encoder [18]. In this paper, we forbid the multiple increases. Given a password in the real vault, we randomly pick an un-increased n-gram to increase; but if all n-grams of the password have been increased, we'll skip the password.

Assume that V_i contains s unique passwords $\{pw_j\}_{j=1}^s$. Let m_j be the number of pw_j in V_i, l_j be the length of pw_j (then the number of n-grams is $l_j - n + 1$), k_j be the number of increased n-grams in pw_j, and m be the size of V_i (i.e., $\sum_j m_j$), respectively. If V_j is real, then $m_j^* = \min\{m_j, l_j - n + 1\}$ n-grams in pw_j are increased in the first step of encoder adjusting and $k_j - m_j^*$ in the second step[6]. The corresponding probability is $f_{\mathrm{N}}(k_j - m_j^*; l_j - n + 1 - m_j^*, p_{\mathrm{i}})$, where $f(k; m, p)$ is the probability mass function of the Binomial distribution $B(m, p)$. Otherwise, k_j increased n-grams in pw_j are all increased in the second step[7]. This probability is $f_{\mathrm{N}}(k_j; l_j - n + 1, p_{\mathrm{i}})$. With $f_{\mathrm{N}}(k_j - m_j^*; l_j - n + 1 - m_j^*, p_{\mathrm{i}})$ and $f_{\mathrm{N}}(k_j; l_j - n + 1, p_{\mathrm{i}})$ as real and decoy probabilities on increased n-gram numbers, we use the ratio of these two probabilities to estimate $\Pr_{\mathrm{DTE}}(DTE^* \mid V_i)$ as

$$p_{\mathrm{AE}}(V_i) = \prod_{j=1}^{s} \frac{f_{\mathrm{N}}(k_j - m_j^*; l_j - n + 1 - m_j^*, p_{\mathrm{i}})}{f_{\mathrm{N}}(k_j; l_j - n + 1, p_{\mathrm{i}})}$$

[6] For simplicity, we exclude the case where an n-gram may appear in multiple different passwords in the vault.

[7] We exclude the case where n-grams in pw_j may appear in the real vault.

$$=\prod_{j=1}^{s}\frac{\binom{l_j-n+1-m_j^*}{k_j-m_j^*}p_{\mathrm{i}}^{k_j-m_j^*}(1-p_{\mathrm{i}})^{l_j-n+1-k_j}}{\binom{l_j-n+1}{k_j}p_{\mathrm{i}}^{k_j}(1-p_{\mathrm{i}})^{l_j-n+1-k_j}}$$

$$=\prod_{j=1}^{s}\frac{1}{p_{\mathrm{i}}^{m_j^*}}\cdot\prod_{t=k_j+1}^{l_j-n+1}\frac{t-m_j^*}{t}. \tag{14}$$

With p_{AE} as the priority function, we propose an *adaptive extra attack*. Note if there exists j with $k_j < m_j^*$ (i.e., $p_{\mathrm{AE}}(V_i)=0$) for a vault, then the vault must be decoy. This is because if this vault is real, then there are at least m_j^* n-grams in pw_j that are increased. As can be seen from this case, exploiting the information leaked by the adaptive encoder, attackers can easily exclude some decoy vaults.

Furthermore, we propose an *adaptive hybrid attack* by combining the adaptive extra attack with the hybrid attack. Its priority function is defined as

$$p_{\mathrm{AH}}(V_i)=\mathrm{sgn}(p_{\mathrm{AE}}(V_i))\cdot p_{\mathrm{H}}(V_i), \tag{15}$$

where sgn is the sign function. At the first attempt, we used $p_{\mathrm{AE}}(V_i)\cdot p_{\mathrm{H}}(V_i)$. But we later found out that its performance (sometimes) was worse than that of the hybrid attack. This may be caused by the estimation error. To optimize its performance, we then use $\mathrm{sgn}(p_{\mathrm{AE}}(V_i))$ for $p_{\mathrm{AH}}(V_i)$ instead of $p_{\mathrm{AE}}(V_i)$. Specifically, the adaptive hybrid attack first excludes the decoy vaults with p_{AE} of 0 and then cracks the remaining vaults by launching the hybrid attack. Thus, the adaptive hybrid attack should always outperform the hybrid attack.

5 Security Evaluation under Our Attacks

We evaluate the existing and our honey vault schemes over the attacks proposed in Section 4 via real-world datasets. The experimental results show that our scheme achieves a significant improvement on security.

5.1 Security Metrics

An attack is more effective if it can use a smaller number of online verifications to identify the real vault for a given ciphertext (see Algorithm 1). This number of online verifications is identical to the rank of the real vault among a large number of decoys in the order defined by the priority function. Thus, we use the *ranks* of real vaults to indicate the security of a honey vault scheme against the attack, as in [13, 14, 18].

Chatterjee et al. [13] and Golla et al. [18] use the *average rank* $\bar{r}$ as a crucial security metric in their evaluation. Chatterjee et al. [13] also define the *accuracy* α in distinguishability. Please note that α is the probability of identifying the real from *only one* decoy (by sorting these two with the priority function), not from a larger number of decoys. To present a comprehensive evaluation, Cheng et al. [14] leverage the *cumulative distribution functions (RCDFs) $F(x)$ of the ranks*. Note that each incorrect master password yields a decoy. Since the master password space is large, it is difficult to calculate the rank of a real vault by generating all decoys. Instead, we choose Cheng et al.'s method [14] to estimate the rank in relative form by sampling N decoys ($N=999$). The rank then is defined as the ratio of the rank to the number of decoys, which is a real number in $[0,1]$ and reflects the relative position in the online verification order. For example, a vault of rank 0.2 will be online verified after checking 20% decoys. In relative form (hereafter, by rank we mean its relative form), $\bar{r}$ and α can be derived from $F(x)$ [14] as

$$\bar{r}=1-\int_0^1 F(x)\,dx, \qquad \alpha=1-\bar{r}. \tag{16}$$

For the sake of comparison fairness, we use the (above) same metrics in our experiments, including $\bar{r}$, α, $F(x)$. In addition, we also use $F(0)$ as in [14], which indicates the proportion of real vaults with *rank 0*—the vaults cracked in *only one-time* online verification (i.e., one guess).

A perfectly secure honey vault scheme guarantees that real and decoy vaults should be indistinguishable, so that the ranks under any attacks follow the uniform distribution $U[0,1]$ (i.e., any attacks perform the same as the randomly guessing attack with a constant priority function). We then have $F(x)=F_U(x)$ ($=x$ for $0\le x\le 1$) and $\bar{r}=\alpha=0.5$. Therefore, we use $F_U(x)$ as the baseline for the comparison.

5.2 Experimental Settings

To present a fair and comprehensive comparison, we utilize the same datasets used in [13, 14, 18]: RockYou as the password dataset and Pastebin as the password vault dataset. RockYou, which is one of the largest leaked plaintext password sets, provides 32.6 million password samples. Being able to maintain the completeness of samples and offer a sufficiently large sample size, it is widely used in the security evaluation on recent password researches [8, 36, 37, 51]. To the best of our knowledge, Pastebin is the only publicly available password vault dataset. it consists of 276 vaults with sizes of 2–50. The data of Pastebin, collected by malware embedded on clients, may indirectly provide us a vision of current exploit means of attackers. In the experiments, we only use these datasets to perform security evaluations. From this perspective, the datasets will bring no harm to valid users and the evaluation results will inspire us to design more secure schemes.

To evaluate the security of honey vault schemes, we do 5-fold cross-validation on Pastebin. Specifically, we randomly divide Pastebin into five parts. We take one part as the test set and the union of other parts as the training set. The vaults in the test set are treated as the real vaults which will be protected (i.e., encrypted) by honey vault schemes and be cracked by attacks. The training set (with RockYou) is used to train the vault models by honey vault schemes. As discussed in Section 4.1, we also use the same training set to train attacks. In this

Table 8: The average rank $\bar{r}$ of real vaults under attacks

Scheme	KL divergence	Single password	Password similarity	Hybrid
Chatterjee et al.'s [13]	14%	10%	22%	6%
Golla et al.'s [18] (static, 10^0)	48%	22%	26%	14%
Golla et al.'s [18] (static, 10^{-1})	37%	19%	23%	14%
Golla et al.'s [18] (static, 10^{-2})	34%	20%	26%	14%
Golla et al.'s [18] (static, 10^{-4})	31%	20%	23%	15%
Golla et al.'s [18] (static, 10^{-6})	30%	19%	24%	14%
Golla et al.'s [18] (static, 10^{-8})	29%	19%	24%	15%
Golla et al.'s [18] (static, 10^{-10})	29%	19%	23%	14%
Golla et al.'s [18] (adaptive, 10^0)	54%	22%	25%	14%
Golla et al.'s [18] (adaptive, 10^{-1})	43%	20%	25%	13%
Golla et al.'s [18] (adaptive, 10^{-2})	40%	21%	25%	13%
Golla et al.'s [18] (adaptive, 10^{-4})	37%	20%	25%	13%
Golla et al.'s [18] (adaptive, 10^{-6})	36%	21%	26%	13%
Golla et al.'s [18] (adaptive, 10^{-8})	35%	20%	24%	12%
Golla et al.'s [18] (adaptive, 10^{-10})	34%	20%	24%	13%
Ours	42%	48%	43%	42%

[1] 10^i represents the pseudocount of Laplace smoothing used in Golla-Markov [18].
[2] The average rank $\bar{r}$ and the accuracy α have the relationship: $\bar{r}+\alpha=1$.

Table 9: The average rank $\bar{r}$ under extra attacks against Golla et al. adaptive schemes [18] with different pseudocounts

Attack	10^0	10^{-1}	10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
Adaptive extra	29%	26%	24%	24%	24%	25%	26%
Adaptive hybrid	9%	8%	8%	8%	7%	7%	7%

setting, we exploit the honey vault schemes to generate decoy and launch attacks to get the rank of each vault in the test set. For each part of Pastebin, we do the above experiment to get ranks of all vaults in Pastebin.

Some important details in the experiments need to be noticed:

1. The honey vault schemes usually need a password dataset to train their single-password model. As in [13, 18], we adopt RockYou for this purpose. This dataset is also used for attacks (if needed).
2. Some attacks need to calculate decoy probabilities (on single password or password feature). This calculation can be launched with the stolen encoders and does not need an extra dataset.
3. Golla et al. [18] do not provide a training method for their reuse-rate approach. For a fair comparison with previous studies, we directly use their parameters in our experiments without training.

5.3 Experimental Results

The performance of our attacks. As shown in Fig. 4, Tables 8 and 9, our proposed attacks perform well against all of the existing schemes. For all the static schemes, the hybrid attack has the best performance, achieving 94% accuracy α ($=1-\bar{r}$) against Chatterjee et al.'s scheme [13] and 85%–86%

Table 10: RCDFs $F(x)$ for honey vault schemes under the corresponding best attacks

Scheme	$F(0)$	$F(1/4)$	$F(1/2)$	$F(3/4)$
Chatterjee et al.'s [13]	55%	93%	98%	99%
Golla et al.'s [18] (static, 10^0)	33%	71%	92%	100%
Golla et al.'s [18] (adaptive, 10^0)	45%	84%	99%	100%
Ours	2%	37%	61%	80%

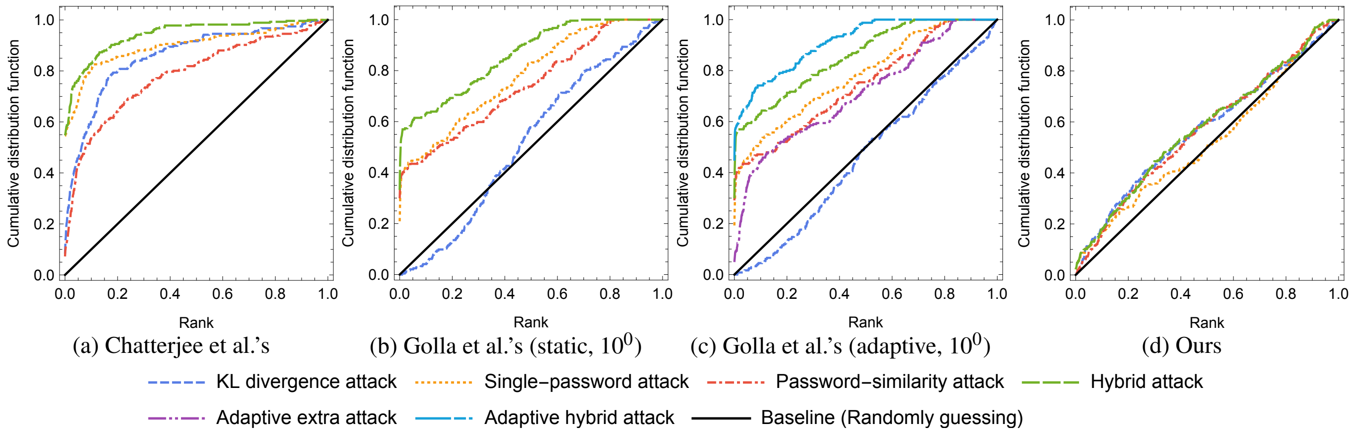
$F(0)$ indicates the cracked proportion of vaults via *only one* online guess.

against Golla et al.'s scheme [18] (with different parameters), as the average rank $\bar{r}$ for Chatterjee et al.'s scheme is 6% and those for Golla et al.'s scheme are 14%–15%. Note Cheng et al.'s honey vault scheme [14] is the same as Golla et al.'s, since we adopt Cheng et al.'s encoder for Golla et al.'s scheme (see Section 2.3). Thus the two schemes achieve the same experimental results, and we do not illustrate the results for Cheng et al.'s scheme separately. With regard to the adaptive scheme, the adaptive hybrid attack outperforms others, capturing 91%–93% accuracy α, as the average ranks $\bar{r}$ are 7%–9%. *Our attacks are based on the theoretically optimal strategy with more accurate estimation, yielding stable and high accuracy.*

The performance of the KL divergence attack is severely affected by the pseudocount of Laplace smoothing used in Golla-Markov. While pseudocount is set to 1, we have the worst attack performance, achieving 46% and 52% accuracy against Golla et al.'s static and adaptive encoders [13], respectively. As shown in Figs. 4b and 4c, the RCDFs are close to the baseline, which means the attack performs close to the randomly guessing attack. We demonstrate that *it is difficult to distinguish real vaults from decoys without any information of the real vault distribution*. The attack, however, only estimates the distance between the vault to be sorted and decoy vaults, not considering the distance between the vault and real vaults. This may lead to some misjudgments. There may exist a target with a "large" distance from the decoy vaults but also with a "larger" distance from the real vaults. The attacker will mistreat the target as the real vault which is actually more likely to be a decoy. To propose more effective attacks, we must exploit both the real and decoy distributions.

The security of the existing schemes. The accuracy of our proposed attacks reveals the vulnerability of the existing schemes. *In terms of the single-password distribution and the password similarity, the existing schemes fail to characterize the real vault distribution.* This is proved by our experimental results, the single-password and the password-similarity attacks achieving 78%–90% and 74%–78% accuracy, respectively. Further, we find out that the pseudocount has an impact on the security of Golla et al.'s schemes [18]: when it is set to 1 these schemes achieve the best security. Here, we only show the RCDFs with this pseudocount.

Exploiting the extra information leaked by the adaptive



(a) Chatterjee et al.'s (b) Golla et al.'s (static, 10^0) (c) Golla et al.'s (adaptive, 10^0) (d) Ours

Figure 4: RCDFs for honey vault schemes under attacks.

scheme, our adaptive hybrid attack increases α to 91%–93%, which is higher than the α on the static schemes with the same pseudocounts under the hybrid attack, i.e., 85%–86%. *The findings of our experiments overturn the conclusion stated in [13]: "adaptive schemes are more secure than the static ones." In fact, an adaptive scheme inevitably does leak information of the encrypted real vault.* By only exploiting the (leaked) information, our adaptive extra attack achieves 71%–76% accuracy without any knowledge of the real vault distribution.

We note that 1) any adaptive scheme (more precisely, its vault model) is adjusted from its original version according to the real vault (which is about to be encrypted by the scheme); 2) the original model has already been trained with a real password vault dataset. If the size of the training set is sufficiently large, adding one more real vault (i.e., the encrypted vault) cannot significantly improve the precision of the model. As shown in Table 8, the average ranks of static and adaptive schemes using the same pseudocount are almost identical under our hybrid attack (without exploiting the leaked information). We thus conclude that *compared with its static variant, an adaptive scheme cannot achieve stronger security and more importantly, the leaked information of the encrypted vault eventually makes it less secure.*

The security of our scheme. We only show the password-similarity attack and the hybrid attack with Feature Overlap in Fig. 4 and Table 8, since the feature performs the best for attacks among the four features demonstrated in Section 4.3. The experimental results for other features are given in Table 16 (see Appendix G).

As shown in Fig. 4 and Table 8, the hybrid attack delivers the best performance, where the average rank $\bar{r}$ and the accuracy α are 42% and 58%, respectively. Compared to the existing schemes with 85%–94% accuracy and 6%–15% average rank, our scheme brings *2.8–7.5 times* cost of online verifications to attackers. Since online verifications can be quickly detected and prevented [17, 21, 42], *our scheme does make a significant improvement on resisting distinguishing attacks in single-leakage case.*

The decreased cracked proportions also illustrate the security improvement. As shown in Table 10, the existing schemes suffer from 33%–55% (i.e., $F(0)$) real vaults cracking via one guess, this value is only 2% for our scheme, which decreases the harm by 93%–96%.

5.4 Further Discussion

Other experiments. We also evaluate the security of honey vault schemes against the password policy attack and the intersection attack (see Appendices F and D). The experimental results are trivial: the attacks can completely break the existing schemes, but are resisted by ours.

Limitation on the dataset. The vault dataset, Pastebin, we used, is not leaked from real vault applications and its size is relatively small (see Section 5.2). Although it is well-studied and used in [13, 14, 18] for security evaluation, the quality of it may yield some bias in our experimental results. Nevertheless, our experiments still demonstrate the insecurity of the existing honey vault schemes [13, 14, 18]. Furthermore, the quality of the dataset does not affect some important conclusions: 1) our construction with the incremental update mechanism can resist intersection attacks; 2) an adaptive scheme leaks extra information of the encrypted vault and is less secure than its static variant.

In general, a dataset with better data quality may help an attacker to more precisely model the real vault/password distribution and more effectively distinguish real and decoy vaults against honey vault schemes (including ours). On the other hand, such a dataset may benefit the design of vault models. Via our generic construction roadmap, using a more accurate multi-similar-password model can generate more plausible-looking decoys and the update security will be maintained.

We note that designing a model and cracking it is not a cat-and-mouse game. If one can design a vault model that precisely captures most of the vaults in the vault space, then arbitrary attackers, even with better-quality datasets, will have little advantage in distinguishing real and decoy vaults.

More powerful attacks. There may be some other information that could be used (as pre-knowledge) to launch attacks. Personal information may be one of the options, since users may create passwords based on name, birthday, phone number, email and username [33, 51]. Attackers may identify a real vault with a higher probability. e.g., by checking if the passwords in the vault match personal information. To resist this type of attack, we may consider using a conditional probability model (e.g., the Personal-PCFG model [33]) that is able to characterize the real vault distribution under the condition of the provided information.

The size of a vault may also provide an extra advantage for attackers in launching online verification. With more accounts on different websites, attackers may be allowed to launch more online verifications on these websites. We do not consider this advantage and leave it as future work.

6 Conclusion and Future Work

We propose a generic construction and further an incremental update mechanism for honey vault schemes. The update mechanism enables the vault scheme to achieve the update security, i.e., *resisting intersection attacks* in the multi-leakage case. We instantiate our scheme with a well-designed multi-similar-password model. Our evaluation with real-world datasets shows that compared with the existing schemes, our instance achieves a significant improvement on security *against (traditional) distinguishing attacks* in the single-leakage case.

Our work may also benefit other research topics. For example, the incremental update mechanism can be used for other HE applications, the multi-similar-password model may benefit password guessing attacks and password strength meters. We leave these as future work.

Acknowledgment

The authors are grateful to the anonymous reviewers and the shepherd, David Freeman, for their invaluable comments that highly improve the completeness of the paper. We also give our special thanks to Qianchen Gu, Zhixiong Zheng, Jiahong Yang, Xiaoxi He and Jiahong Xie for their insightful suggestions and invaluable help. This research is supported by National Key R&D Program of China (2020YFB1805400), National Natural Science Foundation of China (62072010), and European Union's Horizon 2020 research and innovation programme under grant agreement No. 952697 (ASSURED).

References

[1] Have i been pwned? https://haveibeenpwned.com.

[2] How do I update site passwords using auto change? https://support.logmeininc.com/lastpass/help/how-do-i-update-site-passwords-using-auto-change.

[3] How do I view username, password, and note history for sites? https://support.logmeininc.com/lastpass/help/how-do-i-nbsp-view-username-password-and-note-history-for-sites.

[4] LastPass technical whitepaper. https://enterprise.lastpass.com/wp-content/uploads/LastPass-Technical-Whitepaper-3.pdf.

[5] 1Password security design, January 2019. https://1password.com/files/1Password-White-Paper.pdf.

[6] Issue 1930: lastpass: bypassing do_popupregister() leaks credentials from previous site, October 2019. https://bugs.chromium.org/p/project-zero/issues/detail?id=1930.

[7] 1Password. 1password backups. https://support.1password.com/backups/.

[8] Jeremiah Blocki, Ben Harsha, and Samson Zhou. On the economics of offline password cracking. In *IEEE S&P 2018*, pages 35–53.

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

[10] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In *ASIACRYPT 2016*, pages 220–248. Springer.

[11] Joseph Bonneau and Stuart Schechter. Towards reliable storage of 56-bit secrets in human memory. In *USENIX Security 2014*, pages 607–623.

[12] Rahul Chatterjee, Anish Athayle, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. password typos and how to correct them securely. In *IEEE S&P 2016*.

[13] 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.

[14] 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.

[15] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS 2014*, pages 1–15.

[16] Dropbox. File version history overview. https://help.dropbox.com/files-folders/restore-delete/version-history-overview.

[17] David Freeman, Sakshi Jain, Markus Dürmuth, Battista Biggio, and Giorgio Giacinto. Who are you? a statistical approach to measuring user authenticity. In *NDSS 2016*, pages 1–15.

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

[19] Google. Protect your accounts from data breaches with password checkup. https://security.googleblog.com/2019/02/protect-your-accounts-from-data.html.

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

[21] Paul A Grassi, James L Fenton, Elaine M Newton, Ray A Perlner, Andrew R Regenscheid, William E Burr, and Justin P Richer. Nist special publication 800-63b. Digital identity guidelines: Authentication and lifecycle management. *Bericht, NIST*, 2017.

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

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

[24] Ari Juels and Ronald L Rivest. Honeywords: Making password-cracking detectable. In *ACM CCS 2013*, pages 145–160.

[25] Burt Kaliski. PKCS #5: Password-based cryptography specification version 2.0. 2000.

[26] Mathias Karlsson. How I made LastPass give me all your passwords, July 2016. https://labs.detectify.com/2016/07/27/how-i-made-lastpass-give-me-all-your-passwords/.

[27] Jason Kincaid. Dropbox security bug made passwords optional for four hours, June 2011. https://techcrunch.com/2011/06/20/dropbox-security-bug-made-passwords-optional-for-four-hours/.

[28] Eugene F Krause. *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation, 1986.

[29] Russell WF Lai, Christoph Egger, Manuel Reinert, Sherman SM Chow, Matteo Maffei, and Dominique Schröder. Simple password-hardened encryption services. In *USENIX Security 2018*, pages 1405–1421.

[30] Michael Levandowsky and David Winter. Distance between sets. *Nature*, 234(5323):34, 1971.

[31] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[32] Dave Lewis. iCloud data breach: Hacking and celebrity photos, September 2014. https://www.forbes.com/sites/davelewis/2014/09/02/icloud-data-breach-hacking-and-nude-celebrity-photos/.

[33] Yue Li, Haining Wang, and Kun Sun. A study of personal information in human-chosen passwords and its security implications. In *IEEE INFOCOM 2016*, pages 1–9.

[34] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor's new password manager: Security analysis of web-based password managers. In *USENIX Security 2014*, pages 465–479.

[35] 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.

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

[37] William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *USENIX Security 2016*, pages 175–191.

[38] 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.

[39] 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.

[40] 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.

[41] Colin Percival. Stronger key derivation via sequential memory-hard functions. *Self-published*, pages 1–16, 2009.

[42] Benny Pinkas and Tomas Sander. Securing passwords against dictionary attacks. In *ACM CCS 2002*, pages 161–170.

[43] Niels Provos and David Mazieres. A future-adaptable password scheme. In *USENIX ATC 1999*, pages 81–91.

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

[45] Joe Siegrist. LastPass hacked – identified early & resolved, July 2015. https://blog.lastpass.com/2015/06/lastpass-security-notice.html/.

[46] David Silver, Suman Jana, Dan Boneh, Eric Yawei Chen, and Collin Jackson. Password managers: Attacks and defenses. In *USENIX Security 2014*, pages 449–464.

[47] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security 2019*, pages 1556–1571.

[48] Karen Turner. Hacked dropbox login data of 68 million users is now for sale on the dark web, September 2016. https://www.washingtonpost.com/news/the-switch/wp/2016/09/07/hacked-dropbox-data-of-68-million-users-is-now-or-sale-on-the-dark-web/.

[49] Rafael Veras, Christopher Collins, and Julie Thorpe. On the semantic patterns of passwords and their security impact. In *NDSS 2014*, pages 1–16.

[50] Ding Wang, Haibo Cheng, Ping Wang, Jeff Yan, and Xinyi Huang. A security analysis of honeywords. In *NDSS 2018*, pages 1–15.

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

[52] Matt Weir, Sudhir Aggarwal, Breno de Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. In *IEEE S&P 2009*, pages 391–405.

[53] Jeff Yan, Alan Blackwell, Ross Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Secur. Priv.*, 2(5):25–31, 2004.

[54] Peilin Zhao and Steven CH Hoi. Cost-sensitive online active learning with application to malicious URL detection. In *ACM SIGKDD 2013*, pages 919–927.

A Our Single-Similar-Password Model and Its Conditional Encoder

Current single-similar-password models. Wang et al. [51] propose the first single-similar-password model for target password guessing attacks, which exploits a leaked password of a user to guess its another password. Their model characterizes several password-reuse habits, including head or tail modification, capitalization, and leet. Unfortunately, the habits over password reuse cannot be distinctly identified. For example, “Password” can be generated by capitalizing the first character from the string “password”, or it can be generated by modifying the first letter from the same string. This benefits password guessing attacks but is not suitable for HE-based honey vaults. Note that to resist encoding attacks, the encoder of the model needs to parse all generating paths, which yields exponential time complexity. Pal et al. [38] and Pasquini et al. [39] also propose two models, the password-to-path model (pass2path) [38] and the context Wasserstein autoencoder (CWAE) [39], respectively. These two models suffer from the same issue—heavy time complexity.

Our design. For simplicity, we only consider the most popular reuse habit, i.e., head or tail modifications. Specifically, in our model, the new password pw_{new} can be generated from an old one pw_{old} in the following ways:

1. Direct reuse, i.e., $pw_{\text{new}} = pw_{\text{old}}$.
2. Modify the characters in the tail of pw_{old}. The modification operations include deleting, adding and deleting-then-adding. For example, given pw_{old} = “password!”, pw_{new} may be “password” (deleting the last character “!”), “password!*” (adding a character “*”), or “password*@” (deleting “!” and then adding “*@”).
3. Modify the characters in the head of pw_{old} with the same modification operations as above.
4. Modify the head characters and then the tail characters.

In addition, we find that users prefer to directly reuse passwords (i.e., without any modifications) if they already have many passwords. This is intuitive, due to the memory limitation of humans. Therefore, the probability of direct reuse increases as the number of old passwords increases. However, current models [38, 51] only use a static probability for direct reuse, and therefore are not suitable for our construction of the multi-similar-password model. To address this issue, we set an adaptive probability for direct reuse and quantify the probability based on the real-world vault dataset, Pastebin. We find that the proportion of the same password pairs (i.e., direct reuse) in the similar password pairs varies little with vault size. Assuming the proportion is a constant (denoted as

α) independent of vault size (i.e., for different vault sizes, an arbitrary similar pair is a same pair with a constant probability α), we have that the probability of direct reuse is

$$\frac{i \times \alpha}{i \times \alpha + 1 - \alpha},$$

where i is the number of previous passwords (i.e. pw_{new} is the $i+1$-th password). When training, we use the average proportion in the training set for α. Except this probability of direct reuse, other probabilities (e.g. those of modifying characters) are simply counted from the training dataset.

To show the details of character modification in our model, we give an example with "password!" and "password@1" as the old and new passwords. Clearly, "password@1" is generated from "password!" by deleting "!" and adding "@1". Then $\Pr_{\text{SSPM}}(\text{password@1} \mid \text{password!}) = \Pr_{\text{DR}}(\text{False}) \times \Pr_{\text{M}}(\text{Tail}) \Pr_{\text{T}}(\text{Deleting-then-adding}) \Pr_{\text{TDN}}(1) \Pr_{\text{TAN}}(2) \times \Pr_{\text{TAC}}(@) \Pr_{\text{TAC}}(1)$. Here, $\Pr_{\text{DR}}(\text{False})$, $\Pr_{\text{M}}(\text{Tail})$, and $\Pr_{\text{T}}(\text{Deleting-then-adding})$ are the probabilities of not direct reuse, modifying tail, and deleting-then-adding tail characters, respectively; $\Pr_{\text{TDN}}(1)$ and $\Pr_{\text{TAN}}(2)$ are the probabilities of deleting 1 tail character and adding 2 tail characters, respectively; $\Pr_{\text{TAC}}(@)$ and $\Pr_{\text{TAC}}(1)$ are the probabilities of adding characters "@" and "1" to the tail, respectively.

Several details should be carefully considered in our model. Let l_{old} be the length of the old password pw_{old}, l_{HD}, l_{TD}, l_{HA}, and l_{TA} be the character numbers of head deleting, tail deleting, head adding, and tail adding, respectively. Then, the length of the new password pw_{new} is $l_{\text{new}} = l_{\text{old}} - l_{\text{HD}} - l_{\text{TD}} + l_{\text{HA}} + l_{\text{TA}}$ and the longest common substring length l_{LCSStr} of pw_{old} and pw_{new} is $l_{\text{old}} - l_{\text{HD}} - l_{\text{TD}}$. Because $\frac{1}{2} l_{\text{old}} \leq l_{\text{LCSStr}} \leq 2 l_{\text{old}}$, it holds that $l_{\text{HD}} \leq \frac{1}{2} l_{\text{old}}$, $l_{\text{TD}} \leq \frac{1}{2} l_{\text{old}} - l_{\text{HD}}$, $l_{\text{HA}} \leq 2(l_{\text{old}} - l_{\text{HD}} - l_{\text{td}})$, and $l_{\text{TA}} \leq 2(l_{\text{old}} - l_{\text{HD}} - l_{\text{td}}) - l_{\text{HA}}$. Therefore, when calculating $\Pr_{\text{HDN}}$, $\Pr_{\text{TDN}}$, $\Pr_{\text{HAN}}$ and $\Pr_{\text{TAN}}$, all invalid values in the tables should be excluded and meanwhile, the probabilities of the remaining values should be normalized. Note this process is the same as the pruning method [14]. As a result, a decoy seed can be always decoded to a valid vault. Furthermore, if at least one character in the head (or tail) is deleted, then the first head-added (or tail-added) character cannot be the same character as the old one (but other added characters can be identical). This helps us reduce the ambiguity of our model. Similar changes should be applied to $\Pr_{\text{HAC}}$ and $\Pr_{\text{TAC}}$.

With the above designs, our model significantly reduces ambiguity but still cannot eliminate it. This is due to the non-uniqueness of the longest common substring (note the same longest common substrings on different positions are treated as two different ones). For instance, the password "aaaaa" can be modified to "aaaa" in two different ways: deleting the first or the last character. In this case, the probability $\Pr_{\text{SSPM}}(pw_{\text{old}} \mid pw_{\text{new}})$ is defined as the total probability of all modifying methods.

Conditional encoder for our model. We use the method proposed in Section 3.2 to convert this model to a conditional encoder. As discussed above, the conditional encoder needs to parse all the longest common substrings of two passwords. With a generalized suffix tree, this operation can be done in $O(l_1 l_2)$ time, where l_1, l_2 are the lengths of the two passwords, respectively. We note that other operations of the encoder are simple and fast. Thus, our conditional encoder is efficient for real applications.

If one sets our model to characterize more transformation rules of password reuse, then the resulting encoder will suffer from time complexity like others [38, 39, 51]. This is the reason why we prefer to keep our model simple.

B Conditional Encoder for Our Multi-Similar-Password Model

In our multi-similar-password model, the new password pw_{i+1} is generated by reusing $pw_{i'}$ ($1 \leq i' \leq i$) or a brand new selection. This means a valid generating path has the form $(g, r_1, r_2, \dots)$, where g represents the generating path: i' for reusing $pw_{i'}$ and 0 for otherwise. Note that: if $g = 0$, then $(r_k)_k$ must be a generating path in the single-password model; if $g = i'$, then $(r_k)_k$ must be a generating path in the single-similar-password model under the condition of $pw_{i'}$. Using the model-to-encoder transformation proposed in Section 3.2, we can construct the following conditional encoder for our multi-similar-password model.

To encode pw_{i+1} with i given old passwords $(pw_{i'})_{i'=1}^{i}$, the encoder works as

1. Calculate $\frac{1-f(i)}{i} \Pr_{\text{SSPM}}(pw_{i+1} \mid pw_{i'})$ for $1 \leq i' \leq i$ and $f(i) \Pr_{\text{SPM}}(pw_{i+1})$.
2. According to the above probabilities, choose a generating path of pw_{i+1}. If the generating path s is modifying $pw_{i'}$, set $g_i = i'$, otherwise, set $g_i = 0$.
3. Encode g_i by the IS-DTE for the distribution of g_i (the probability is $f(i)$ for 0 and $\frac{1-f(i)}{i}$ for 1 to i).
4. If $g_i = 0$, encode pw_{i+1} (the remaining rules in generating path s) by the encoder (done by Cheng et al.'s transformation [14]) for our single-password model, otherwise, encode pw_{i+1} by the conditional encoder (made by our extended transformation) for our single-similar-password model with the given old password pw_{g_i}.
5. Concatenate these seeds, pad the concatenation to a fixed-length seed with random bits, and output the seed.

C Leakage Detection Mechanisms

Leakage detection for honey vaults. Our mechanism, making good use of honeypot accounts, is "simple and natural". To implement this for a vault, the vault application needs:

1. Generate honeypot accounts, which should be indistinguishable from the real accounts by attackers and will not

be misused by the user. This generation can be done by a well-designed algorithm or the user himself. Note by an account, we here mean a triple consisting of a username, a password, and a domain, e.g., (Alice07, 12345678, yahoo.com).

2. Register these accounts on the corresponding websites and store them in the (real) vault. The websites of honeypot accounts should provide login reminders. Specifically, the user will be noticed if anyone attempts to log in to these accounts. Many real-world websites, e.g., Google, satisfy this requirement. Note that the registration is carried out on the user's device under his consent, which does not violate the websites' terms of service[8]. In addition, these accounts may get disabled by the websites due to inactivity. To keep the accounts active, the vault applications need to log in to these accounts periodically (on the user's device).
3. Set up leakage reminders, which will be triggered if the websites of honeypot accounts send any login reminders (except those yielded by the application's periodical logins). The vault applications can provide an online service for the reminders. Specifically, the online service monitors the login reminders from these websites, and send a vault leakage reminder to the user if a login reminder occurs.

If the user gets this vault leakage reminder, he should change all his passwords stored in the vault and the master password immediately. Furthermore, he should never reuse these passwords. Note some vault applications support automatic update for the website passwords, e.g., LastPass [2], which offers great convenience.

With the mechanism, the attacker who steals the honey vault file will not distinguish the honeypot accounts. But to verify the correctness of the decrypted vaults, the attacker will try to the accounts stored in the vaults. Since the websites usually block the accounts with too many failed login attempts, the attacker will log in to as many accounts as possible. Therefore, the attacker provably logs in to honeypot accounts and further a vault leakage reminder will be triggered.

The main challenge of the design is to generate plausible-looking honeypot accounts, including choosing appropriate websites, picking up usernames and generating passwords. A straightforward way is to require users to generate honeypot accounts on their own. But this compromises usability. Instead, we may design a generation algorithm for honeypot accounts via the following considerations:

1. *Websites.* To prevent attackers from telling honeypot accounts, the websites for honeypot accounts should look as "normal" as those for real accounts. A potential way is to choose websites according to their popularity. Specifically, given a list of website candidates (providing login reminders), the generation algorithm can choose one with a probability that is proportional to its user number. In this way, a more popular website (with more users) has a greater probability to be chosen for honeypot accounts. Intuitively, the choice seems to be made by a real user (human).
   When a user visits a honeypot-account website, he may misuse the account (e.g., via the auto-fill or auto-login function provided by the vault applications), and further trigger a false alarm of the vault file leakage. To prevent the misuse, the algorithm can choose the websites where the user rarely visits for his honeypot accounts (note that these websites may still be popular and have many other visitors). Furthermore, the logo and domain of the website can be attached to the leakage reminder, so that the user can check if the reminder is falsely triggered by himself. For instance, the notice may be in the form: "If you just logged in Yahoo as Alice07, please ignore this notice."
2. *Usernames.* The usernames of honeypot accounts should look like those of the user's real accounts. Otherwise, attackers may tell the honeypot accounts from real ones. To satisfy this requirement, the algorithm can directly reuse the usernames of real accounts in the vault.
3. *Passwords.* The passwords of honeypot accounts should look consistent with other passwords in the vault. A potential generation method is to sample a password from our multi-similar-password model with the passwords of real accounts.

Another consideration we should take into the detection design is to *avoid false alarms of vault file leakages*. An attacker who does not steal the vault file may intentionally launch denial-of-service (DoS) attacks to trigger false alarms. Specifically, the attacker may know the user's real username (usually is the username for honeypot accounts) and then try to log in on many websites with this username. Note that if the attacker encounters a honeypot account, the login attempt will trigger a login reminder, and further a false alarm of leakages. Then the user will be requested to change all the passwords in the vault. Fortunately, the attacker has to pay a high price to succeed, since there are a considerable amount of websites that need to be tried online. In addition, this attack may trigger malicious login reminders for real accounts. When the reminders occur, the DoS attack can be reliably noticeable.

Password breach alerts. Password breaches provide attackers side information of password vaults and enable them to offline distinguish real and decoy vaults. This is an important and practical threat for honey vaults, but is not considered in previous researches [13, 14, 18]. Fortunately, we can leverage the leakage detection mechanism for honey vaults and existing alert mechanisms for password breaches to address this threat.

[8]The registration may violate Facebook's terms of service, because Facebook prohibits the registration with fake personal information or multiple registrations from one user.

Some websites may leverage security mechanisms (e.g., honeywords [24]) to detect the leakage of password files and actively notice users. In addition, several real-world services, e.g, HaveIBeenPwned [1] and Google [19, 47], have been deployed to monitor the password breaches, and further provide breach alerts for password vault applications. Using the password breach alerts, the vault applications can identify the leakage of passwords stored in a vault. If a password is detected to be leaked, the user should change it along with all other similar passwords (note the user should not reuse these passwords ever again). Then the vault application needs to fully (not incrementally) update the vault and erase all backups containing these passwords.

In this way, we can effectively prevent an attacker from stealing both a vault file and a password in the vault:

1. If the password is leaked first, the user may get an alert of the password breach. Then he will change the password and other similar passwords, fully (not incrementally) update the vault and erase all the backups containing these passwords. Therefore, the attacker who steals the updated vault later cannot offline distinguish the real and decoy vaults.
2. If the vault file is leaked first, the user also may get a reminder. Then he will change all passwords in the vault and the master password. Hence, the leaked vault file is useless to the attacker who gets a password in it later.

This method may not completely prevent an attacker from stealing both the vault and password, but it significantly reduces the probability of this happening to mitigate the threat.

Malicious website detection. Malicious websites may be built by attackers to steal passwords from users, which may bring the same threat for honey vaults as password breaches. A malicious website may pretend to be a normal one (e.g., Yahoo) to confuse users and steal their passwords. This may be easily prevented by vault applications with the auto-fill feature. But a malicious website also may entice users to register on it and get their passwords.

Fortunately, we can use some existing mechanisms (e.g., malicious URL detection [54]) to prevent users from accessing malicious websites. Registering to malicious websites should be prohibited. But if a user insists to do so, we recommend him to use a randomly-generated password and store the password in plaintext (without encryption). In this way, the leakage of this password will bring no advantage to attackers in distinguishing real and decoy vaults, and other passwords will remain safe.

D Intersection Attacks Against Honey Vault Schemes

We evaluate the security of honey vault schemes against intersection attacks with the real-world datasets.

Table 11: The performance for honey vault schemes under the trivial intersection attack

Scheme	$\bar{r}$	α
Chatterjee et al.'s [13]	0%	100%
Golla et al.'s [18] (static, 10^0)	0%	100%
Golla et al.'s [18] (adaptive, 10^0)	0%	100%
Ours	50%	50%

Table 12: The average rank $\bar{r}$ for single-password models under the single-password attack

Model	Chatterjee-PCFG	Golla-Markov	Weir-PCFG	Neural network	Best-Markov[1]
$\bar{r}$	18%	35%	33%	40%	50%

[1] Best-Markov is the 5-order Markov model using distribution-based normalization and Laplace smoothing with the pseudocount of 0.001.

Experimental settings. For each vault (with a size larger than 2) in the vault dataset Pastebin, we randomly shuffle the passwords in the vault and treat the last password as a new added one. In this way, we get the old and new versions for each (real) vault (here, the old version is the vault without the last password). Then we use the honey vault schemes to generate the decoys for these real vaults (note the decoys have respective two versions as well). The rest of the experiment settings are the same as those in Section 5.

Intersection attacks. We leverage a trivial intersection attack for the evaluation. This attack only leverages the similarity between the old and new versions of vaults, but not considers the difference between the real vault distribution and the vault model. For each candidate vault V_i with its old and new versions $V_i^{\mathrm{o}}, V_i^{\mathrm{n}}$, the priority function $p_{\mathrm{TIA}}(V_i)$ of the trivial intersection attack is equal to 1 if V_i^{o} is the same as V_i^{n} except for the last password, otherwise, 0. In other words, the attack directly excludes these vaults of which two versions are not similar.

Experimental results. As shown in Table 11, the intersection attack can directly tell the real vault with 100% accuracy for all existing honey vault schemes. This is because the two versions of each decoy vault are randomly generated and there is only a very small probability that the two versions are similar. In contrast, our scheme generates the new version of each decoy vault by adding a new password to the old version. Therefore, it can resist the intersection attack.

E Evaluating Single-Password Models

To instantiate our construction with a good single-password model, we evaluate the existing models with the single-password attack.

Existing single-password models. We here evaluate Chatterjee-PCFG [13], Golla-Markov [18], Weir-PCFG [52],

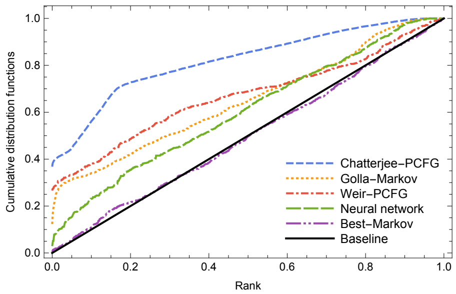


Figure 5: RCDFs for different models trained with RockYou.

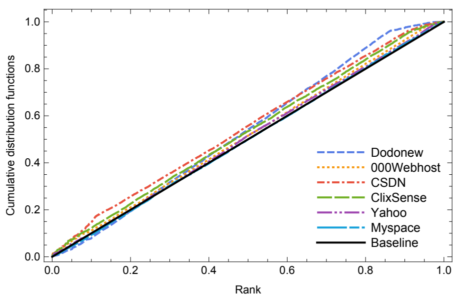


Figure 6: RCDFs for Best-Markov trained with different datasets.

the neural network model [37], and Markov models [36] using different methods. The neural network model proposed by Melicher et al. [37] is the same as 10-order Markov model except that the transition probabilities are calculated by recurrent neural networks. The performance of the model heavily depends on its parameter settings. Here we use the default settings in the example configuration on GitHub. Similar to the neural network model, the performance of Markov models relies on its normalization and smoothing methods. We note that in the Markov models with distribution-based normalization, we train independent Markov models for passwords with different lengths, which is not considered in [36] but [18]. We employ this operation because it can make a significant improvement on the performance of Markov models. So, we only present the experiments with this operation.

Experimental settings. We randomly choose 50% passwords from RockYou as the training set for PMTEs and attacks, while randomly selecting 10^4 passwords from the remaining part of RockYou as real passwords. Apart from that, other experimental settings are the same as those for honey vaults.

Experimental results. As shown in Fig. 5, Tables 13 and 12, the 5-order Markov models using distribution-based normalization and Laplace smooth with pseudocount of 10^{-3} can guarantee the expected security, where the average rank is 50%. We denote these models as *Best-Markov*. We also achieve good performance for Best-Markov via using other password datasets including Dodonew, 000Webhost, CSDN, ClixSense, Yahoo, and Myspace. Note these datasets are extensively used in password researches, e.g., [36,50,52]. As shown

Table 13: The average rank $\bar{r}$ for Markov model under the single-password attack

Normalization		End-symbol			Distribution-based		
Order		3	4	5	3	4	5
Pseudocount	10^0	31%	35%	39%	33%	37%	39%
	10^{-1}	30%	37%	43%	34%	38%	44%
	10^{-2}	31%	36%	44%	33%	40%	48%
	10^{-3}	32%	35%	42%	33%	39%	**50%**
	10^{-4}	30%	39%	45%	34%	42%	48%
	10^{-5}	32%	36%	46%	32%	42%	47%

Table 14: The average rank $\bar{r}$ for Best-Markov trained with different datasets under the single-password attack

Dataset	Dodonew	000Webhost	CSDN	ClixSense	Yahoo	Myspace
$\bar{r}$	47%	48%	45%	47%	49%	50%

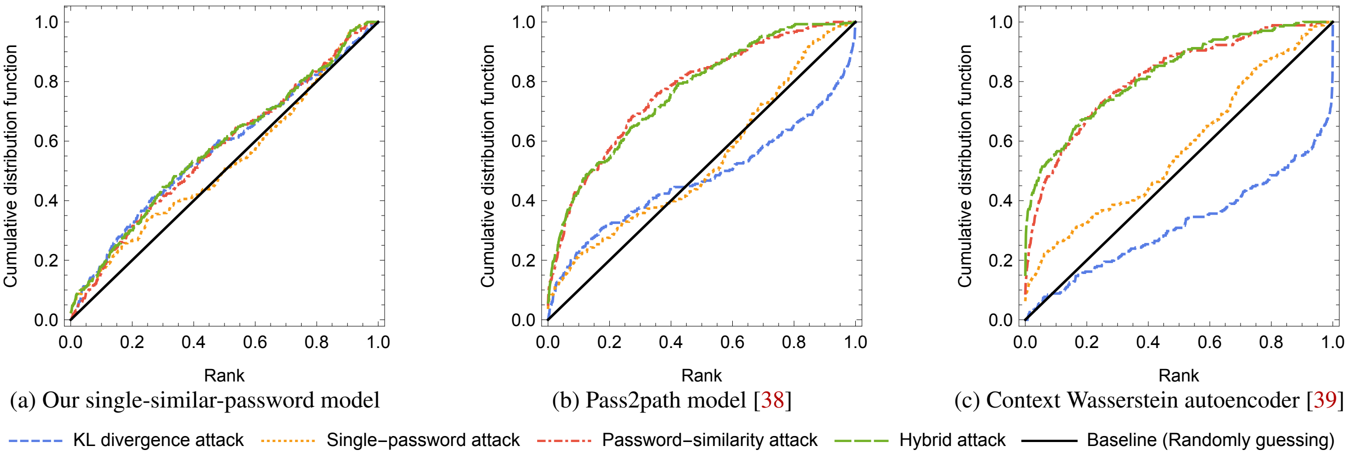
in Fig. 6 and Table 14, the average ranks of Best-Markov all approach to the expected value ($\approx$ 50%) and meanwhile, the RCDFs are all close to the baseline. Due to the good performance of Best-Markov, we will use it in our vault model.

F Evaluating by Password Policy Attack

Since a vault may contain multiple or even hundreds of passwords with different policies, the password policy attack may be a serious threat to existing honey vault schemes. We originally planned to launch the password policy attack on the real vault dataset. But we find some websites (e.g., Google) have changed their policies, and some passwords in the real vaults do not comply with the current password policies. Since we do not know the old policies when the passwords were registered, we use artificially-made policies to evaluate the security of honey vault schemes against the password policy attack. Specifically, we define three types of password policies:

1. Password length is not less than n ($n = 6, 8$). We denote this policy as nL.
2. Password contains at least n ($n = 2, 3$) types of characters in lower-case letters, upper-case letters, digit numbers, and special characters. We denote this policy as nC.
3. A combination of the above policies, denoted as n_1Ln_2C.

Table 15 shows the proportion of passwords complying with the password policies in all passwords generated by the single-password model of existing vault models [13, 18]. Informally, if one password in a vault has such a policy, then the (policy) attack can exclude many decoys and only leave this proportion of vaults for online verification. The stronger the policy is, the more effective the password policy attack presents. More interestingly, if several passwords have such policies, the left proportion of the vaults is the product of the proportion for each password. So for a vault of a large size, the attacker probably can directly tell the real vault, i.e., completely breaking the honey vault schemes. Fortunately, our vault model can adjust itself and ensure that all passwords



(a) Our single-similar-password model (b) Pass2path model [38] (c) Context Wasserstein autoencoder [39]

Figure 7: RCDFs for our honey vault scheme using different single-similar-password models under attacks (with the best performance similar meter)

Table 15: The proportion of passwords complying with the password policy

Policy	6L	8L	2C	3C
Chatterjee-PCFG	91%	33%	23%	2%
Golla-Markov	95%	50%	49%	20%
Our scheme	100%	100%	100%	100%
Policy	6L2C	6L3C	8L2C	8L3C
Chatterjee-PCFG	23%	2%	11%	1%
Golla-Markov	48%	19%	30%	14%
Our scheme	100%	100%	100%	100%

generated by it comply with the policies.

G Evaluating Single-Similar-Password Models

To choose a single-similar-password model for our multi-similar-password model, we evaluate two existing models, pass2path [38] and CWAE [39] along with our simple model (Appendix A) under our proposed attacks. We do not consider Wang et al.'s model [51] because: 1) pass2path always outperforms this model on target password guessing [38]; 2) Wang et al. do not open-source the code, which brings difficulty in comparison.

Experimental settings. The settings are the same as those in Section 5, except the following.

1. *Training.* We leverage pass2path and CWAE which are trained by the authors and provided on GitHub. We do not train these two models with the real dataset Pastebin, because the dataset is too small for them and cannot provide sufficient training. We note that our model is simple and can be trained with a small-scale dataset.
2. *Password-similarity features.* Pass2path is training from the password pairs (pw_1, pw_2) satisfying that the Levenshtein distance between pw_1 and pw_2 is not more than 5. So we use this feature as Feature M in the password-similarity attack. As for CWAE, we still use Feature LCSStr as Feature M. The rest of the attack settings are the same as those for our model in Section 5.
3. *Encoder.* As we discussed in Appendix A, pass2path and CWAE yield heavy time complexity in encoding. Therefore, we do not implement their encoders. Instead, we directly sample passwords from these models to generate decoy vaults in the experiments.

Table 16: The average rank $\bar{r}$ for our honey vault scheme with different single-similar-password models

Attack	Ours	Password-to-path model [38]	Context Wasserstein autoencoder [39]
KL divergence	42%	52%	68%
Single password	48%	47%	42%
Password similarity (Levenshtein)	45%	24%	23%
Password similarity (LCS)	47%	31%	**18%**
Password similarity (Manhattan)	46%	**23%**	24%
Password similarity (Overlap)	**43%**	37%	22%
Hybrid (Levenshtein)	44%	26%	22%
Hybrid (LCS)	46%	28%	19%
Hybrid (Manhattan)	43%	**24%**	22%
Hybrid (Overlap)	**42%**	35%	**17%**

Experimental results. As shown in Fig. 7 and Table 16, our simple design performs the best: the average rank $\bar{r}$ is not less than 42% under any attacks. This means no attacks can achieve more than 58% accuracy in distinguishing real and decoy vaults. Both pass2path and CWAE cannot generate plausible-looking decoys: targeting pass2path, the password similarity attack with the Manhattan meter achieves 77% accuracy; targeting CWAE, the hybrid attack with the Overlap meter achieves 83% accuracy. Since our simple model performs well on decoy generating, it may also perform well on password guessing. We leave this as future work.