How to Design Honey Vault Schemes

Chensheng Zhang(✉), Tingwei Fan and Jingwei Jiang

College of Cyber Science, Nankai University, Tianjin 300350, China
zhang.chen.sheng@mail.nankai.edu.cn

Abstract. A password vault encrypts and stores a user's multiple passwords in a vault, enabling only to remember the master password. A honey vault is a specific type of password vault that yields plausible-looking decoy vaults when the master password is incorrectly guessed. This forces attackers to shift from offline guessings to online verifications. A number of honey vault schemes have been proposed, yet most of the existing schemes have not considered the behavior of attackers in practical scenarios. Accordingly, we provide a system model and a new security metric to capture the attacker's abilities.

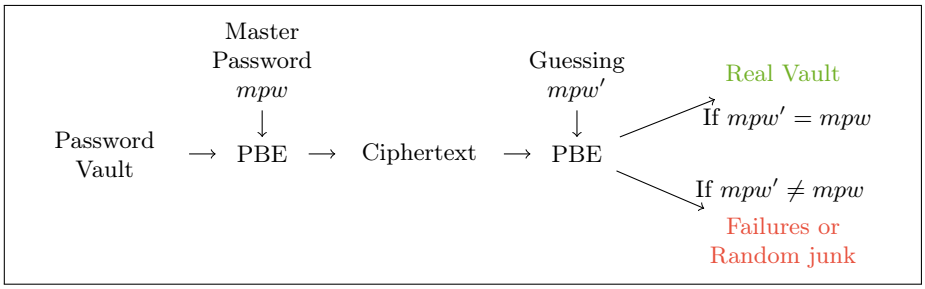
When a user registers, a website only allows passwords meeting specific requirements, we call this *password creation policies*. We reveal that the attacker can use password creation policies to distinguish honey vaults, which brings significant advantages to the attacker. Experimental results show that checking only five passwords can exclude 90% honey vaults. To resist this policy verification attack, we propose that passwords following different creation policies must be processed with different distribution-transforming encoders (DTEs), rather than one common DTE as in previous schemes. To meet this demand, we provide an algorithm that can construct ideal DTEs for any determined password distribution. We believe this work provides new feasible directions for DTE, and contributes to a better understanding of honey vault.

Keywords: Honey vault, Policy verification attack, Honey encryption.

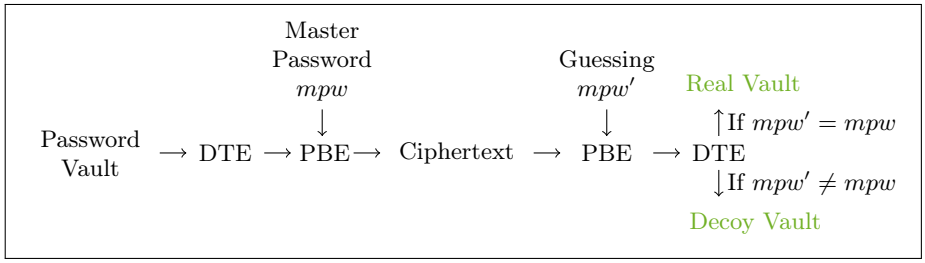
1 Introduction

Passwords are the most prevalent user authentication, and may also be the only cryptographic keys that ordinary users can keep in mind [22]. Despite numerous security risks and usability flaws (e.g., guessing [19, 33, 35], phishing [24, 31], and memorization [10]), as well as various alternatives (e.g., graphical [2] and biometric [36] authentications), passwords remain irreplaceable in the foreseeable future due to their simplicity, low cost, and ease of change [5, 6, 37].

As the number of accounts owned by a user increases, users find it hard to remember a vast number of passwords. Surveys show that ordinary Internet users have 80-107 different online accounts [12, 23], while human brains can only remember 5-7 passwords [17]. This inevitably leads to behaviors like using popular passwords (e.g., `password` and `qwerty`), and password reuse across sites, making user passwords vulnerable to credential stuffing/tweaking attacks (see [3, 32,34]). To mitigate threats posed by remembering multiple passwords, password vaults (i.e., password managers), where a user only needs to remember the master password, are recommended by experts in both academia and industry [9, 21].



(a): Traditional PBE-based password vault



(b): Honey password vault

Fig. 1: Subfigures (a) and (b) show the difference between traditional and honey password vaults in the view of attackers.

A conventional password vault utilizes a user's chosen master password to encrypt (i.e., password-based encryption, PBE) and store users' passwords. Once the server is breached, attackers can conduct offline guessing attacks against a user's master password to decrypt the password vault. Particularly, when an incorrect candidate is tried, the vault will be decrypted as random junks instead of semantically meaningful messages, signaling an incorrect guess. Worse still, attackers can utilize existing password cracking softwares (e.g., Hashcat [30] and JtR [25]) to facilitate offline attacks, and conduct over 10^{12} guesses per day. All this reveals the threats posed by offline brute-force attacks.

To defend against such vulnerabilities, honey encryption (HE) [16] is utilized in password vaults [11, 26] to build honey vault (HV) schemes. *The core idea is to generate plausible-looking decoy vaults when incorrect master passwords are tried.* To do this, a HV scheme utilizes the natural language encoder (NLE) that comprises a distribution transforming encoder (DTE). When HE is implemented, the DTE first randomly encodes messages (e.g., passwords in a vault) from a known distribution (modeled by a password model, e.g., PCFG [35] and Markov [19]) into seeds, then encrypts them with a symmetric encryption algorithm (e.g., AES [28]). The decryption is the reverse. In this way, as shown in Fig. 1, *only* the correct master password can decrypt the real password vault, and incorrect ones lead to decoy vaults. This protects password vaults against offline brute-force attacks, and forces an attacker to use methods such as online verifications

(which can be more effectively mitigated, see [4, 13, 14, 18]) to find a decrypted vault is the user's real one.

1.1 Challenges and Motivations

Since NoCrack [26] was proposed in 2015, several schemes [7, 8, 11] have been proposed, yet few studies have paid attention to the behavior of attackers in practical scenarios. There is also no summary of HV design routines. Without a systematic methodology, convincing security is unlikely, and there is no basis for HV schemes to be put into practical application. Specifically, *the following key research questions (**RQs**) remain to be answered:*
RQ1: In the HV scheme, what causes security design against one attack to be *completely ineffective* in another new attack. Researchers usually focus more on adopting new technologies and how to use these technologies to improve previous schemes. However, if this issue is not thoroughly studied, more newly proposed schemes will only become part of the cycle *break-fix-break-fix* , which cannot provide effective promotion for research of HV design.
RQ2: In HV schemes, the domains and usernames are stored in plaintext because attackers can easily verify their validity. On this basis, how much information can attackers obtain from this to distinguish honey vaults?
RQ3: Is there an easy to understand method to construct ideal DTEs?

1.2 Our Contribution

A summary of existing honey vault attackers. We first propose a new system model that divides honey vault schemes into (1) natural language encoders (NLEs) and (2) the encryption algorithm. Existing attacks *all* fall into these two categories. More specifically, Golla et al.'s K-L divergence attack [11], Cheng et al.'s encoding attack and distinguishing attack [8] exploit vulnerability in NLEs, while Cheng et al.'s intersection attack and Rao et al.'s master password reset attack [27] exploits vulnerabilities in encryption algorithms.
Password policy verification attacks. We propose a policy verification attack that exploits inconsistencies in password creation policies of sites stored in the password vault. Through experiments, we reveale that attackers can use sites' policies to significantly distinguish honey vault, and, *for the first time*, provide an attack instance that honey vaults are getting more vulnerable with the increased size of each vault. As a countermeasure, we suggest designing different DTEs for sites with different creation policies to ensure that attackers cannot obtain any additional information from policies.
Construct ideal DTE. We first utilized the principle of random oracle to provide an algorithm that can obtain an ideal DTE when the password distribution of the website is given. Experiments show that by using such a DTE, attackers can only rank real vault around 50%, which is ideal security.
Some insights. We provide two new insights: 1) When using encryption algorithm in designing security scheme, it's necessary to consider existing attacks against relevant algorithms and carefully check whether the assumptions required for these attacks can be implemented in the scenario of the scheme. User's

demand for the correctness of decryption may break some security natures of encryption algorithm. By conducting such confirmation, potential attacks can be identified as much as possible, rather than being found in a flash, which will greatly prevent the scheme from being applied to reality (e.g. honey vault has not been put into practice yet). 2) When using a simulation setup in experiments to test attacker's ability, it is necessary to carefully consider the attacker's behavior in real scenarios and confirm the information known to the attacker. Otherwise, there may be a trap that the simulation setup accidentally reduces the attacker's available known information, leading to a weakening of the attacker's ability, and causing inaccurate evaluation of the security.

2 Backgroud

2.1 Related Work

Honey encryption. At EUROCRYPT'14, Juels and Ristenpart proposed honey encryption (HE) [16] to resist brute-force attacks by yielding plausible-looking messages for incorrect keys. Since attackers cannot locally identify the correctness of decrypted messages, HE is resistant to offline brute-force attacks as revealed in [7, 8, 11, 15, 16, 26]. At a high level, a honey encryption scheme is comprised of two parts: (1) a natural language encoder (NLE) that contains a password model (e.g., PCFG in [26] and Markov in [11]) and a distribution transforming encoder (DTE), and (2) a symmetric encryption scheme(e.g., AES [28]). In particular, a password model is used to model password distributions in vaults, DTE contains a randomized encoder and a deterministic decoder to encode the password model into seeds, and SE encrypts this seed to a ciphertext.

Existing honey vault scheme. At IEEE S&P'15, Chatterjee et al. [26] introduced a honey vault scheme *NoCrack* based on Honey Encryption (HE). At CCS'16, Golla et al. [11] proposed adaptive encoders that adjust themselves according to the encrypted vault to make decoys more similar to it. At USENIX SEC'19, Cheng et al. [8] found both Chatterjee et al.'s [26] and Golla et al.'s [11] encoders suffer from encoding attacks, they further proposed a generic transformation that can convert a probability model to an encoder resisting encoding attacks. Later, at USENIX SEC'21, Cheng et al. [7] found honey vault schemes suffer from intersection attacks and they further proposed an incremental update mechanism to resist the intersection attacks. Recently, Rao et al. [27] found honey vault schemes suffer from master password reset attacks.

Each new scheme adds a new design on top of the previous scheme to resist new attacks. This is because the underlying principles of each attack are different, so the added resistance design can be combined like building blocks. We give a more detailed explanation in Sec 3.2.

Policy attack. As far as we know, Cheng et al.'s policy attacks [7] at USENIX'21 may be the closest to our policy verification attacks. They used artificially-made policies to make password policy attacks. Specifically, they defined three types of password policies: 1) Password length not less than n (e.g., n = 6,8) is denoted 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, denoted as nC;

3) Combination, denoted as n_1Ln_2C. They experimented with single password under eight policies: $6L, 8L, 2C, 3C, 6L2C, 6L3C, 8L2C, 8L3C$.

There are two issues need to be pointed out. First, there are different types of password policies in the Pastebin dataset such as: limiting the maximum length of the password, a minimum length of 4 or 5, containing lower-case letters/upper-case letters/digit numbers/special characters. These policies cannot be fully covered by the above eight policies. Second, their scheme uses a conditional encoder, which means that the encoding and decoding of each password are affected by previous passwords in the same vault. There is no relevant experiment or explanation on whether passwords based on multiple different policies included in a vault will affect their scheme's resistance to policy attack or not.

Different from existing practices, We will use real-world policies. Specifically, we will use policies stemming from the password requirements during website registration to perform a policy verification attack in Sec 4.

2.2 Preliminary

Notation. We use $y \leftarrow_{\$} A(x)$ to denote running randomized algorithm A on input x and setting y equal to its output. If instead A is deterministic we write $y \leftarrow A(x)$. If G is a game we let $\Pr[\mathrm{G} \Rightarrow \text{true }]$ denote the probability that G outputs true. Let $\mathcal{S}$ be a set, a distribution on $\mathcal{S}$ is a function $p : \mathcal{S} \rightarrow [0, 1]$ such that $\sum_{s\in\mathcal{S}} p(s) = 1$. The maximum probability ω of a distribution p is defined to be $\omega = \max_{s\in\mathcal{S}} p(s)$. By $s \leftarrow_p \mathcal{S}$ we denote sampling an element $s \in \mathcal{S}$ according to the distribution p. That is, each $s \in \mathcal{S}$ is chosen with probability $p(s)$.

Distribution-transforming encoder (DTE). A distribution-transforming-encoder (DTE) is a pair of algorithms DTE = (encode, decode) defined relative to a message space $\mathcal{M}$ and a set S called the seed space. Via $S \leftarrow$ encode (M) the algorithm encode takes a message $M \in \mathcal{M}$ as input and outputs a seed $S \in \mathcal{S}$. A DTE must satisfy correctness, that for any message $M \in \mathcal{M}, \Pr[\mathrm{decode}(\mathrm{encode}(M)) = M] = 1$.

As a preprocessing before encryption, DTE plays a role in expanding the plaintext space. Although there may be multiple plaintexts corresponding to the same password in this expanded plaintext space, it does indeed expand the plaintext space for encryption, that is to say, correspondingly expanding the key space. Therefore, the intersection of this expanded key space and the original key space (master password space) may contain more elements, so that the adversary will receive multiple decoys and can't directly determine which is the real one.

An important detail is that the same input of DTE may correspond to different outputs. The reason is as above, and previous articles have directly followed this property without any explanation.

Hash functions. A hash function is a function $\mathbf{H} : \{0, 1\}^* \rightarrow \{0, 1\}^n$ which maps strings of arbitrary length to strings of some fixed length n. The length n will always be clear from context. In this work, we model the hash function as a random oracle. As shown in in Fig. 2, random oracle can achieve true uniform randomness and have the same output value for the same input.

Random Oracle (RO)
Start with an empty table containing two columns,
one for input and the other for output.
If input X has not appeared in the input column,
RO uniformly and randomly selects O(X),
outputs O(X) and records it in the table;
If X has appeared before, output the corresponding O(X).

Fig. 2: How a Random Oracle works

HEnc(K, M)
$S \leftarrow_{\$} encode(M)$
$R \leftarrow_{\$} \{0,1\}^n$
$C_2 \leftarrow_{\$} \mathrm{H}(R \| K) \oplus S$
Return (R, C_2)

HDec(K, C)
$(R, C_2) \leftarrow C$
$S \leftarrow \mathrm{H}(R \| K) \oplus C_2$
$M \leftarrow \mathrm{decode}(S)$
Return M

Fig. 3: J-R's DTE-Then-Encrypt Construction

DTE-then-encrypt. Juels and Ristenpart introduce a framework for constructing honey encryption schemes [15, 16]. As shown in Fig. 3, this construction first uses DTE = (encode, decode) to preprocess the message, then uses the hash function to process symmetric encryption. Inspired by this construction and how random oracles work, we use a simulation algorithm to experiment on honey vaults in Sec 4.4 and Sec 5.2.

3 Our Honey Vault Model

3.1 Application scenario and necessary properties

In the honey vault scenario, the user submits a master password and several passwords to the server, and then the server encrypts and stores passwords for safekeeping. The server will not save the master password. Afterward, users can obtain other passwords through the master password. In practice, the domains and usernames are stored in plaintext. This is because a real username is registered on the domain, but its decoy is not. Thus, the attacker can easily identify the decoy vaults by confirming the domains and registering with the usernames. To the best of our knowledge, there does not exist an effective solution to hide domains and usernames. We leave this as an open problem. In the following text, we only consider encrypting passwords when discussing encryption.

We list the following properties as the foundation of our system model.

Property 1: Scheme has correctness, i.e., using the correct master password can definitely decrypt correctly. This property is designed to ensure the usability of the scheme, and most encryption algorithms can meet this property. Algorithms with a probability of failure in decryption, such as code-based encryption algorithms cannot be used.

Property 2: Master password is set by the user and needs to be memorized by user. This property is an objective issue. Real vault will inevitably appear in the attacker's sight by using advanced password guessing techniques for guessing master passwords. In the foreseeing future, this situation will not only remain, but also become more severe for ordinary users.

Property 3: Using the wrong key to decrypt ciphertext will result in incorrect decoy vault which are indistinguishable from real vault. This property indicates

the core goal of the honey vault scheme. A necessary condition of property 3 is that each password in these decoy vaults appears to be correct. More specifically, each password must meet the corresponding website's password creation policy. This is the foundation of our policy verification attack in Sec 4.

3.2 Threat model

We first explain some basic abilities that attackers possess. Attackers know all the details of the encryption algorithm and know all the randomness used. Attackers can implement all functions of the server, including decryption. If an attacker obtains the leaked data of encrypted vaults, he can provide decryption services. *The only missing information for attackers is the master password.*

Following our model, We are the first to divide the analysis of attackers into two categories: *encryption algorithms* and *natural language encoders*.

We define *encryption algorithm attackers* as those who exploit existing attacks on encryption algorithms to find honey vault vulnerabilities. This kind of attackers tries to get side information of the user's master password from encryption algorithms. They confirm existing attacks one by one (by consulting technical books or relevant literature) to determine the assumptions used for certain attacks. Next, they transform these assumptions into scenarios in the honey vault. Finally, they need to determine whether these scenarios can occur in reality, and if so, they can implement a certain attack. Their attacks are related to users' behavior and require additional data leakage.

There have been examples of encryption algorithm attackers in the past. Cheng et al. [7] used the sensitivity of block cipher to insertion and modification to propose the intersection attack. Rao et al. [27] used the meet-in-the-middle attack to propose the master password reset attack. They did not provide any explanation about the encryption algorithm itself, but instead set up realistic scenarios and conducted experiments to demonstrate. If there is no user's password behavior (modifying password or resetting master password), it is almost impossible to successfully make ciphertext collision, i.e. the attacker cannot even obtain one collision. Now, through user's behavior, at least one collision is guaranteed because of property 1 *scheme has correctness*. And according to the security property of the encryption algorithm, it is very likely that there is only this one. So the attacker can recognize it as real one.

Combining user's behavior and existing attacks against encryption algorithms may find new realistic scenarios for attacking honey vault. This approach can also be applied to other systems that use encryption algorithms. We will leave it for future work.

We define *natural language encoder attackers* as those who exploit vulnerabilities in NLEs to distinguish honey vaults and identify all true passwords. This kind of attackers tries to get side information from DTE itself. They do not assume users' behavior or additional data leakage. They will pay attention to the input-output distribution of DTE and look for imbalances in it. There has been examples of such attackers in the past. Golla et al. [11] used Kullback-Leibler divergence to measure the differences in the generated distribution of passwords and rank the correct vault into the 1.31%(smaller rank means better attack).

Cheng et al. [8] used the vulnerability of adaptive NLEs that its output space did not fully cover decoy seeds to achieve 99% accuracy in verifying real vault.

We point out attackers can make more attacks from password guessing. In password guessing, passwords are unequal, meaning that attackers should already have a ranking of master passwords. However, due to the lack of master password data in Pastebin, previous studies have erased this ranking and assumed that all master passwords are equal. This weakens the attacker's ability since, in reality, attackers can add the probability of the master password to the sorting function when evaluating honey vaults.

Another point should be taken care is that: it is usually assumed that attackers have the ability to exhaust all possible master passwords offline and the real vault will definitely appear in the attacker's sight with many decoy vaults. According to our property 2, this assumption is correct. But is this assumption really as beneficial to the attacker as it seems? We have a explanation as follows.

3.3 New metric

The goal of attackers is to make a successful online attack. *Cracking honey vaults offline is an intermediate process.* Attackers only have a small number of verifications per website, and the total number will slowly increase as the vault size increases. When attackers' failed online logins reach this total number, online verification cannot continue, indicating that attackers have failed.

Based on this observation, we propose a new security metric: *the number of active honey vaults.* Active honey vaults refer to those honey vaults that can be verified online by attackers in a priority order over the real vault. Vividly, these vaults are active in preventing attackers since they can indeed trap attackers. Fig. 4 shows the relationship between our new metric and rank, which was used as the metric for evaluating security in previous research. A key issue that should not be ignored is: *To ensure the real vault is indeed among these candidate vaults, how many guesses about the master password must be made ?*

This issue has been ignored in the past: Previous researchers usually set several (e.g. 1000) candidate vaults, including one real vault in experiment setup. If the attacker can determine that the real vault is among 1000 honey vaults, then under ideal security conditions, real vault would rank at 500, but this does not necessarily mean that it cannot be breached. We still use the parameters in previous experiments for discussion. When the vault size is 200, attackers only make three online login verifications on each website to find the real one, which is realistic; When the size is beyond 1000 (such as 2000 in previous experiments), attackers can fully confirm the authenticity of all honey vaults by logging into each website once, in this case, honey vaults will not be effective.

In practice, guessing the master password is similar to trawling guessing attack because both attackers have stored data about passwords. The difference is each data possessed by the trawling guessing attacker can correspond to a unique password, while the ciphertext in the honey vault scheme can be decrypted into many honey vaults by different guessing. Using password guessing techniques in guessing the master password remains an open task.

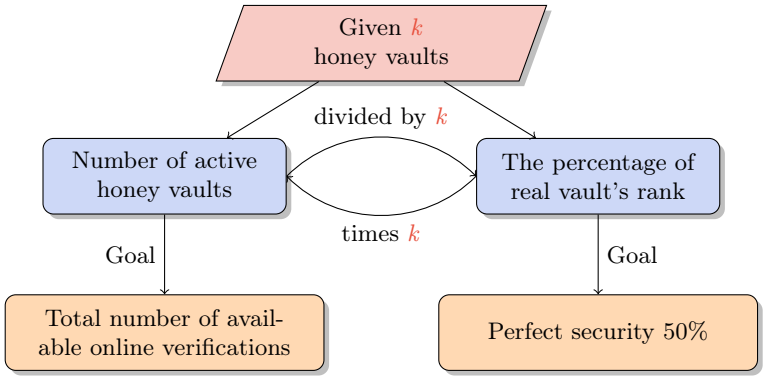


Fig. 4: Our new metric *number of active honey vault* aims to prevent attackers from *online* attacking by wasting all available online verifications with active honey vaults. Previous metric *The percentage of real vault's rank* aims to make attackers distinguish honey vault *offline* like flipping a coin.

4 Policy verification attack

Password creation policies for websites can be easily obtained by attackers. If there is a password in honey vault does not meet the corresponding website's policy, then attackers can confirm the vault as false. In this way, attackers can exclude these honey vaults offline, resulting in an increase of attacker's ability. We show the effectiveness of policy verification attack through the NoCrack.

4.1 The dataset

We state that Pastebin is the only one real-world leaked plaintext password vault dataset. Table 1 shows statistics data of the Pastebin dataset.

Table 1: Overview of Pastebin dataset, stats of vaults only consider size > 1

Stat Item	**Count Number**
Websites	1354
Websites don't found policy	1127
Policy types	80
Passwords	2669
Passwords have found policy	1054
Passwords don't meet policies	381
Vaults containing at least one password found policy	**236**
Vaults complying with corresponding password creation policies	**101**
Vaults containing at least one password don't meet policies	135

Pastebin is leaked from 2011. It is so old that many websites no longer exist, or have updated policies, resulting in a mismatch between vault passwords and current policies. For websites that no longer exist, attackers do not have the ability to attack them online. So we set the corresponding password to non-existent, meaning that such a password does not provide any policy verification information. For cases that passwords do not match the policies currently used by the website, policy verification attack will exclude the true vault. Table 1 shows there are still 381 passwords do not meet the corresponding website's current policies, although we have made every effort to collect relevant policies

Table 2: Different size vaults in policy verification attack experiment

Size	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	18	19	23
Number	29	16	13	7	2	8	4	2	3	2	2	2	1	2	1	2	1	2

4.2 Experiment with Pastebin

We emphasize that there is no data for master passwords in the Pastebin dataset, so we follow the approach in previous articles: using the Rockyou dataset to sample master passwords. RockYou dataset contains 32.6 million passwords, is one of the largest leaked plaintext password sets. As shown in Table 1, we experiment with **101** vaults than can make policy verification attacks. Table 2 shows the size of these vaults. Results are shown in Fig. 5.

We present the overall framework of the experiment as follows.

1.Preparatory stage: Randomly sample k master passwords from the Rockyou dataset. One of these k master passwords is used as the real one to encrypt the vault. Then, we decrypt the ciphertext with these k master passwords to obtain honey vaults.

2.Attack stage: Attackers only do one thing for these honey vaults: check whether each password meets the corresponding website's creation policy. If a honey vault contains a password that does not meet the policy, it will be excluded. We use *check number* to represent the number of website policies that have been checked. We use *remain* to represent the number of honey vaults that have not been excluded after each check.

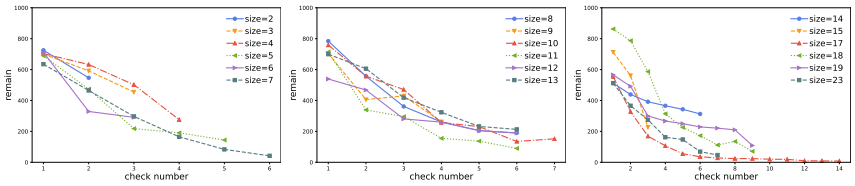
3.Statistical stage: For each vault, record the change in its *remain* as the *check number* increases. We experiment with each vault three times and take the average value as the result of this vault. For a size with only one vault, use the result of that vault as the size result. For a size with multiple vaults, use the average of these vaults' results as the size result.

4.3 Result analysis

The dash curve in Fig. 5 shows that checking only 3 to 5 passwords can bring good effects(50% - 90%) for excluding honey vaults.

It can also be observed from Fig. 5 that there are differences in the excluding rates of different policies. We further experiment with this matter, and the comparison in Fig. 6 shows that complex policies have better exclusion effects. Since we only care about exclusion efficiency, the vaults our experiment with are selected from **236** vault containing at least one password found policy.

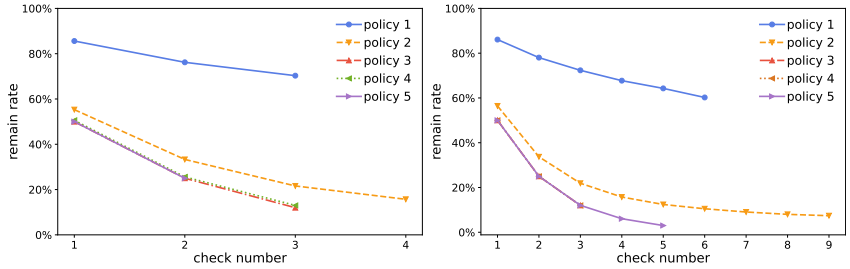
Fig. 5 also shows that: the more passwords in the vault, the more vulnerable to our policy verification attacks. The core idea is that: *no one password can satisfy all password policies. The behavior of generating decoys without considering the creation policies will have a non-negligible probability that the honey vault containing at least one incorrect password, which can be recognized by our policy verification attacks.* This probability increases as the number of passwords increases, ultimately result in a high probability. We have provided a theoretical analysis of the relationship between vault size and security in the Appendix A.



(a) k=1000 (b) k=1000 (c) k=1000

(d) k=10000 (e) k=10000 (f) k=10000

Fig. 5: Results of policy verification attack for vaults that comply with corresponding password creation policies. The *remain* decreases as the check number increases, indicating that there are fewer potential candidate vaults in attacker's view. Thus, attackers are more likely to identify real one.



(a) Experiment with size =49 vault, this vault already check these five types policies each three or more times during policy verification attack

(b) Experiment with different vaults. These vaults already check one of these five types policies three or more times during policy verification attack

Fig. 6: We experiment with six different policies. blue line: At least 6 characters; yellow line: At least 8 characters; red line: At leats 8 characters, contain numbers, letters and special characters; green line: 8-20 characters, contain letter and number; purple line: At least 8 characters, contain lowercase letter, uppercase letter and number; brown line: 8-64 characters, contain lowercase letter, uppercase letter and number . The difference in results indicates: The more restrictions on policies, the better effect on policy verification attack. This is because complex policies can narrow down the set of available passwords, making it smaller probability for honey vaults to generate decoys within that set.

Remark. Policy verification is an ordinary operation that requires no complex or advanced knowledge (such as statistics, machine learning, etc.). However, it can bring such a great advantage to attackers, which implys that the website's password creation policy contains more information than commonly believed.

4.4 More experiments using simulation algorithms.

In the past, experiments were conducted by sampling passwords from several websites' leaked datasets to work as vaults, but this did not meet the requirements of policy verification attacks because users' vaults should be composed of passwords from many different websites. In order to be intuitive and focus only on the core idea, we conducted experiments using simulation algorithms.

Let X be the set of passwords, K be the set of master passwords, Y be the set of ciphertexts, P is a probability distribution defined in X and defaulted to uniform distribution unless otherwise specified. Algorithms Enc and Dec share a record table T, and details are as follows.

Algorithm Enc : Given input $k \in K, pw \in X$, search records that look like $Enc(k, pw) = c$ in T If T has records like this, select a c as the output; If there is no such record in T, randomly select a c from Y that no record $Dec(k, c)$ in T, add records of $Enc(k, pw) = c$ and $Dec(k, c) = pw$ to T, and output c;

Algorithm Dec : Given input $k \in K, c \in Y$, search record that looks like $Dec(k, c) = pw$ in T. If T have a record like this, take pw as the output; If there is no such record in T, select a $pw \leftarrow_P X$, add records of $Enc(k, pw) = c$ and $Dec(k, c) = pw$ to T, and output pw;

We present the overall framework of the experiment as follows.

1.Preparatory stage: Let X be the set of passwords. Let X_1 be the set of passwords which follow creation policy of website 1, Let X_2 be the set of passwords which follow creation policy of website 2. We set X_1 and X_2 have the same number of passwords and satisfy $X = X_1 \cup X_2$. The password generation process first has $b \leftarrow_\$ \{1, 2\}$, then randomly selects a password in X_b and adds a marker b to the selected password. A vault containing n passwords is generated through n password generation processes.

2.Encryption stage: Each vault has a master password $k \leftarrow_\$ K$. Encrypts each vault to obtain ciphertexts. The encryption process involves using the master password to sequentially Enc each password in the vault.

3.Attack stage: Attackers decrypt ciphertexts to obtain honey vaults and sort them. During decryption, the ciphertext is sequentially Dec using the master password in K. Calculate weights $Pr = \prod_{i=1}^{n} Pr(pw_i) Pr(pw_i \in X_i)$ for each honey vault and then sort them in descending order of weight. Randomly determine the order for honey vaults with the same weight.

4.Statistical stage: Count the average rank of each vault. A smaller rank means a better attack.

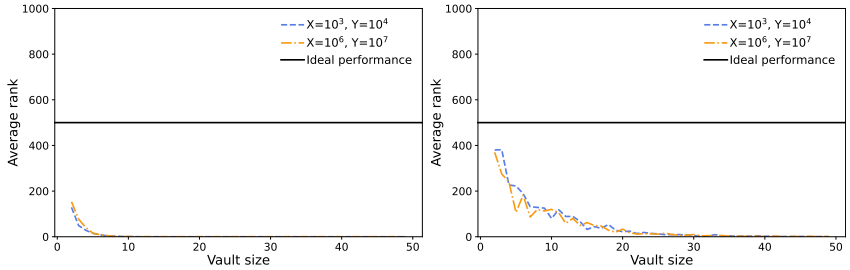
There are two types of relationships between different password creation policies: (a) There is no intersection between them. We call them *mutually exclusive*

policies. (b)There is overlap between them. We call them *partially overlapping policies*. We experiment with both types.

We will set 20 vaults for each size to reduce the impact of accidentally breaking through a vault. Following the experiment setups from prior studies, set 1000 as the number of candidate master passwords and vault size $n \in [2, 49]$.

(a) Mutually exclusive policies. Let X_1 and X_2 both contain 50% passwords in X and satify $X_1 \cap X_2 = \emptyset$. We experiment with two sets of parameters, one is $|X| = 10^3$, $|Y| = 10^4$ and the other is $|X| = 10^6$, $|Y| = 10^7$. These two sets of parameters serve as controls to demonstrate that the sizes X and Y do not affect the desired expression results. Results are shown in Fig. 7(a)

(b) Partially overlapping policies. Let X_1 and X_2 both contain 90% passwords in X and their intersection contain 80% passwords in X. We still set two experiments using parameters $|X| = 10^3$, $|Y| = 10^4$ and $|X| = 10^6$, $|Y| = 10^7$. Results are shown in Fig. 7(b).



(a) Mutually exclusive policies (b) Partially overlapping policies

Fig. 7: Results of Simulation Experiments of policy verification attack. As the size increases, attackers can check more passwords to better exclude honey vaults and ultimately exclude all honey vaults to confirm the real one.

Analysis. Even though policies have a high overlapping rate, honey vault is still breached. This warns us to use different DTEs for different policies to eliminate the additional information of password creation policies.

5 Ideal DTE

The honey vault scheme could provide randomly generated passwords, but users may still generate passwords by themselves. These human-chosen passwords are likely to be non-uniform. When designing DTEs for these passwords, the decoy passwords in the honey vault should have the same non-uniform distribution.

5.1 Theoretical analysis

Let X be the set of passwords, K be the set of master passwords, c is the ciphertext, and Enc is the encryption of the honey vault scheme. The advantage

that an attacker $\mathcal{A}$ can gain from honey vault is defined by

$$\mathrm{Adv}_{\mathrm{HV}} = |\Pr[\textbf{Use honey vault} \Rightarrow \mathit{true}] - \Pr[\textbf{Trival guess} \Rightarrow \mathit{true}]| .$$

In Fig. 8, **Use honey vault** means attackers exhaust all the master passwords in K to gain $|K|$ honey vaults, then try to find the real one. **Trival guess** means attackers try to find the real one from $|K|$ candidates based on password distribution. Security of the honey vault is reflected in whether honey vaults will leak more information than password distribution.

We need to point out, the perfect security is that candidates in the view of attackers are the same as directly sampling from the password distribution. Attackers can reverse strategy to have an advantage more than trivial guesses if

Trival guess
$pw \leftarrow_p \mathrm{X}$
$pw_i \leftarrow_p \mathrm{X}, 1 \le i \le \lvert K\rvert - 1$
$c = \{pw, pw_i\}, 1 \le i \le \lvert K\rvert - 1$
$pw' \leftarrow \mathcal{A}(c)$
Return $pw' = pw$

Use honey vault
$pw \leftarrow_p \mathrm{X}$
$k \leftarrow_{\$} \mathrm{K}$
$c \leftarrow \mathrm{Enc}(pw, k)$
$pw' \leftarrow \mathcal{A}(c)$
Return $pw' = pw$

Fig. 8: Games define advantages of honey vault attackers.

$$\Pr[\textbf{Use honey vault} \Rightarrow \mathit{true}] \le \Pr[\textbf{Trival guess} \Rightarrow \mathit{true}].$$

Specifically, attackers can arrange sorting of candidates in reverse order.

The key is output distribution when attackers decrypt the ciphertext. If this distribution is indistinguishable from the password distribution, then $\mathrm{Adv}_{\mathrm{HV}}$ is negligible. Otherwise, attackers have a non-negligible advantage and completely break through the honey vault. We demonstrate this threat through experiments.

5.2 Simulation experiment for DTE

In the experiment setup, we exclude interference from other factors to ensure that attackers only use the differences between password distribution and DTE to distinguish honey vaults. We present the overall framework as follows.

1.Preparatory stage: Let X be the set of passwords. The password generation process is selecting a password $pw \leftarrow_{P_1} X$. A vault containing n passwords is generated through n password generation processes.

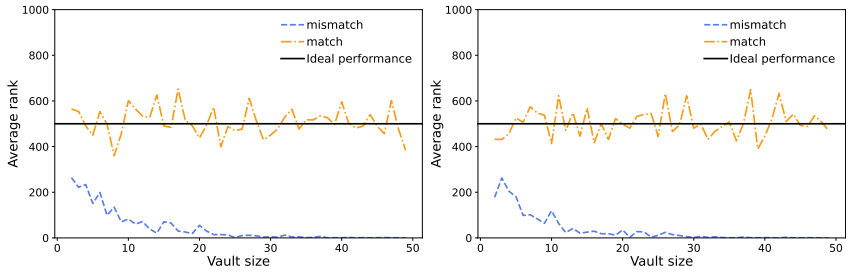
2.Encryption stage: For each positive integer $n \in [2, 49]$, set 20 vaults. Each vault has a master password $k \leftarrow_{\$} K$. Encrypts each vault to obtain the corresponding ciphertext. The encryption process involves using the master password to sequentially *Enc* each password in the vault.

3.Attack stage: The attacker decrypts the ciphertext to obtain honey vaults and sorts them. During decryption, the ciphertext is sequentially *Dec* using the master password. Calculate weights $Pr = \prod_{i=1}^{n} Pr(pw_i)$ for each honey vault and then sort them in descending order of weight. Randomly determine the order for honey vaults with the same weight.

4.Statistical stage: Count the average rank of each vault. A smaller rank means a better attack.

We experiment with two cases : *match* and *mismatch*.

Experiment of match: Set the password distribution P_1 to half of X has an 80% probability, while the remaining half has a 20% probability. The distribution P of Enc and Dec maintains a uniform distribution. It should be noted that a new empty table T needs to be set. Results are shown in Fig. 9.
Experiment of mismatch: Set the password distribution P_1 to half of X has an 80% probability, while the remaining half has a 20% probability. Set the distribution P of Enc and Dec is the same distribution as P_1. A new empty table T should also be set. Results are also shown in Fig. 9.



(a) $|X| = 10^3, |Y| = 10^4$ (b) $|X| = 10^6, |Y| = 10^7$

Fig. 9: The yellow line (*match*) fluctuates around 50%, maintaining as the vault size increases. the blue line (*mismatch*) sharply decreases as the size increases. This comparison shows that DTE should match the password distribution. Two sets of parameters serve as controls to demonstrate that the size of X and Y do not affect the desired expression results

.

Analysis. The difference in Fig. 9 between *match* and *mismatch* is because attackers can gain a non-negligible advantage from creation policies, and this advantage will stack up as the vault size increases.

5.3 Construct ideal DTE

We propose a new theoretical approach for designing DTEs: using simulation algorithms to determine the encode and decode tables of DTEs. We give specific algorithm processes in **Algorithm:** Generate encode and decode table for DTE.

We explain three details. 1) The plaintext set is larger than the password set. This is a realistic condition. 2) For each master password and plaintext, *decode* has a definite record. For each password and master password, *encode* may have several different records; randomly select one of them as the output of *encode*. This reflects that DTE can encode a password into multiple results. 3) The master password set in reality may not be deterministic, but it can be fixed using a hash function. This approach does not pose security risks, as HE is different from traditional encryption scenarios which aim to avoid situations where plaintext is seen by attackers. As a countermeasure, HE allows attackers to see a large number of decoys, confusing them not knowing which one is true. Attackers directly exhaust the output space of the hash function for the attack, just like exhausting the master password.

Algorithm: `Generate encode and decode table for DTE`

Input: The passwords set $X = \{pw_1, pw_2, \cdots, pw_{|X|}\}$, the passwords probability distribution P on X, A master password set $K = \{k_1, k_2, \cdots, k_{|K|}\}$. A plaintext (ciphertext) set $Y = \{y_1, y_2, \cdots, y_{|Y|}\}$ for encryption algorithm. An empty table T, consisting of two columns: *encode* and *decode*.

Output: A table T, consisting of two columns: *encode* and *decode*

for $i \leftarrow 1$ **to** $|X|$ **do**
  for $j \leftarrow 1$ **to** $|K|$ **do**
    select a $y \leftarrow_{\$} Y$.
    If there have no record $decode(y, k_j)$ in T, add $encode(pw_i, k_j) = y$ to *encode*, add $decode(y, k_j) = pw_i$ to *decode*. j=j+1.
    else j=j
  i=i+1.
for $i \leftarrow 1$ **to** $|Y|$ **do**
  for $j \leftarrow 1$ **to** $|K|$ **do**
    If there have a record $decode(y_i, k_j)$ in T , j=j+1.
    else select a $pw \leftarrow_P X$, add $encode(pw, k_j) = y_i$ to *encode*, add $decode(y_i, k_j) = pw$ to *decode*, j=j+1.
  i=i+1.
return table T

Remark. The results in Fig. 9 suggest that DTE and encryption algorithms should lean towards imbalance and be consistent with the vault distribution as same as possible. There is almost no research on imbalanced algorithms. This is because traditional encryption required balanced algorithms, and attacks on related algorithms are all attempts to find algorithms' imbalanced parts. Therefore, the academic community generally believes that imbalanced algorithms are meaningless. We point out that imbalanced encryption algorithms might have potential usage scenarios in honey vault, furthermore, the honey encryption.

6 Further discussion

Setting up decoy accounts. There are two ways to set up decoy accounts. 1) if the wrong master password is entered, the honey vault includes real accounts and decoy accounts. As long as the attacker fails to log in once, it can be confirmed that this is a decoy vault. Regardless of whether the failure occurred in the bait account or in the real account, the number of wasted online verification is one (no alert will be triggered upon successful log in). In this case, decoy accounts did not provide better security. 2) honey vault includes wrong passwords/accounts even if the correct master password is entered. There is no practical and feasible solution for this approach yet, we will leave it for future work.

Resisting policy verification attack. As we discussed in Sec 5.2, DTE should be consistent with the password distribution. To achieve this, a necessary information is the website's creation policy. Honey vault sever need to keep track of

password creation policies for the websites of users and modify the DTE to only generate compliant passwords. This is a long-term and complex task beacuse users will constantly add new websites, and stored websites may modify policies. We emphasize that this is something sever must do since they are required to provide security services to users. There already have relevant work now [1, 20].

Resisting intersection attack. Intersection attack [7] assumes that after the user changes the password in the vault, attackers can obtain ciphertext from both old and new versions. Vaults obtained by decrypting these two versions of ciphertext with the true master password are the same except for the password that has been changed. But their corresponding decoy vaults are difficult to achieve this. This is because block ciphertext algorithms are sensitive to changes in plaintext, and a partial change in plaintext may cause all parts of the ciphertext to change. Cheng et al. [7] note that AES in CTR-mode merged with PBKDF satisfies the prefix-keeping property that: if two plaintexts have same prefix, then the ciphertext obtained by encrypting them with any key will have the same prefix and if two ciphertexts have the same prefix, they will always obtain plaintext with the same prefix after being decrypted by any (wrong) key. Based on this property, they proposed an update mechanism to make vault change always occur at the last one position. When adding a password, add it to the last one position. When deleting a password, mark the password as deleted (in plaintext) without changing the ciphertext. When modifying a password, first mark the old one as deleted and then add the new one to the last position. However, *this mechanism will increase the number of ciphertexts when users modify passwords, and the ciphertext of deleted passwords will be permanently stored.*

We point out that the encryption algorithm of stream ciphers can be used in a honey vault scheme to resist intersection attacks without these drawbacks. Stream cipher algorithms naturally have incremental feathers that changing any part of the plaintext will only cause the corresponding part of the ciphertext to change, while the other parts will remain unchanged. When users modify passwords, no additional operation is required, and the number of ciphertexts to be stored remains unchanged. When a user deletes a password, mark it as deleted, and then when the user adds a new password, directly replace the deleted password. Ciphertexts of the deleted password will not be permanently stored.

7 Conclusion

To the best of our knowledge, this work is the first to give a system model for honey vaults and explains this puzzle: *In the previous honey vault schemes, the security setting to resist one attack was completely ineffective when facing new attacks.* We also propose a password (creation) policy verification attack, and our experiments reveal that it's necessary to use different DTEs for different password creation policies. In order to construct a large number of DTEs, we give an algorithm to generate DTE for any password distribution. We hope our work can provide a more comprehensive understanding of honey vault and promote its further application in reality.

References

1. Alroomi, S., Li, F.: Measuring website password creation policies at scale. In: Proc. ACM CCS 2023. pp. 3108—-3122
2. Biddle, R., Chiasson, S., Oorschot, P.V.: Graphical passwords: Learning from the first twelve years. ACM Comput. Surv. 44(4), No.19 (2012)
3. Bijeeta, P., Tal, D., Rahul, C., Thomas, R.: Beyond credential stuffing: Password similarity models using neural networks. In: Proc. IEEE S&P 2019
4. Blocki, J., Zhang, W.: DALock: password distribution-aware throttling. In: Proc. PETS 2022. pp. 516–537
5. Bonneau, J., Herley, C., van Oorschot, P., Stajano, F.: Passwords and the evolution of imperfect authentication. Commun. ACM 58(7), 78–87 (2015)
6. Bonneau, J., Herley, C., Van Oorschot, P.C., Stajano, F.: The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In: Proc. IEEE S&P 2012. pp. 553–567
7. Cheng, H., Li, W., Wang, P., Chu, C.H., Liang, K.: Incrementally updateable honey password vaults. In: Proc. USENIX SEC 2021. pp. 857–874
8. Cheng, H., Zheng, Z., Li, W., Wang, P., Chu, C.H.: Probability model transforming encoders against encoding attacks. In: Proc. USENIX SEC 2019. pp. 1573–1590
9. Clifford, C., Sharon, P.: Keep Your Passwords Strong and Secure With These 9 Rules (May 2022), https://www.cnet.com/tech/mobile/keep-your-passwords-strong-and-secure-with-these-9-rules/
10. Gao, X., Yang, Y., Liu, C., Mitropoulos, C., Lindqvist, J., Oulasvirta, A.: Forgetting of passwords: Ecological theory and data. In: Proc. USENIX SEC 2018
11. Golla, M., Beuscher, B., Dürmuth, M.: On the security of cracking-resistant password vaults. In: Proc. ACM CCS 2016. pp. 1230–1241
12. Hanamsagar, A., Woo, S.S., Kanich, C., Mirkovic, J.: Leveraging semantic transformation to investigate password habits and their causes. In: Proc. ACM CHI 2018. pp. 1–12
13. Herley, C., Schechter, S.E.: Distinguishing attacks from legitimate authentication traffic at scale. In: Proc. NDSS 2019. pp. 1–13
14. Islam, M., Bohuk, M.S., Chung, P., Ristenpart, T., Chatterjee, R.: Araña: Discovering and characterizing password guessing attacks in practice. In: Proc. USENIX SEC 2023. pp. 1867–1884
15. Jaeger, J., Ristenpart, T., Tang, Q.: Honey encryption beyond message recovery security. In: Proc. EUROCRYPT 2016. pp. 758–788
16. Juels, A., Ristenpart, T.: Honey encryption: Security beyond the brute-force bound. In: Proc. EUROCRYPT 2014. pp. 293–310
17. Keith, M., Shao, B., Steinbart, P.J.: The usability of passphrases for authentication: An empirical field study. Int. J. Hum. Comput. Stud. 65(1), 17–28 (2007)
18. Lu, B., Zhang, X., Ling, Z., Zhang, Y., Lin, Z.: A measurement study of authentication rate-limiting mechanisms of modern websites. In: Proc. ACSAC 2018
19. Ma, J., Yang, W., Luo, M., Li, N.: A study of probabilistic password models. In: Proc. IEEE S&P 2014. pp. 689–704
20. Moritz, H., Mario, S., Johannes, B., Johannes, B.: Password requirements markup language. In: Proc. ACISP 2016. pp. 426–439
21. Oesch, S., Ruoti, S.: That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers. In: Proc. USENIX SEC 2020. pp. 2165–2182

22. O'Gorman, L.: Comparing passwords, tokens, and biometrics for user authentication. Proc of the IEEE 91(12), 2021–2040 (2003)
23. Pearman, S., Thomas, J., Naeini, P.E., Habib, H., Bauer, L., Christin, N., Cranor, L.F., Egelman, S., Forget, A.: Let's go in for a closer look: Observing passwords in their natural habitat. In: Proc. ACM CCS 2017. pp. 295–310
24. Peng, P., Xu, C., Quinn, L., Hu, H., Viswanath, B., Wang, G.: What happens after you leak your password: Understanding credential sharing on phishing sites. In: Proc. ACM ASIACCS 2019. pp. 181–192
25. Peslyak, A.: John the Ripper password cracker, `http://www.openwall.com/john/`
26. Rahul, C., Joseph, B., Ari, J., Thomas, R.: Cracking-resistant password vaults using natural language encoders. In: Proc. IEEE S&P 2015. pp. 481–498
27. Rao, T., Su, Y., Xu, P., Zheng, Y., Wang, W., Jin, H.: You reset I attack! a master password guessing attack against honey password vaults. In: Proc. ESORICS 2023
28. Rijmen, V., Daemen, J.: Advanced encryption standard. Federal Inf. Process. Stds. (NIST FIPS) 19, 22 (2001)
29. Shannon, C.E.: Communication theory of secrecy systems. The Bell System Technical Journal 28(4), 656–715 (1949)
30. Steube, J.: Hashcat (2018), `https://hashcat.net/hashcat/`
31. Thomas, K., Li, F., Zand, A., Barrett, J., Ranieri, J., Invernizzi, L., Markov, Y., Comanescu, O., Eranti, V., Moscicki, A., et al.: Data breaches, phishing, or malware? Understanding the risks of stolen credentials. In: Proc. ACM CCS 2017
32. Wang, D., Zou, Y., Xiao, Y.A., Ma, S., Chen, X.: PASS2EDIT: A Multi-step generative model for guessing edited passwords. In: Proc. USENIX SEC 2023
33. Wang, D., Zou, Y., Zhang, Z., Xiu, K.: Password guessing using random forest. In: Proc. USENIX SEC 2023. pp. 965–982
34. Wang, K.C., Reiter, M.K.: Detecting stuffing of a user's credentials at her own accounts. In: Proc. USENIX SEC 2020. pp. 2201–2218
35. Weir, M., Aggarwal, S., de Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: Proc. IEEE S&P 2009. pp. 391–405
36. Yang, Y., Lu, H., Liu, J.K., Weng, J., Zhang, Y., Zhou, J.: Credential wrapping: From anonymous password authentication to anonymous biometric authentication. In: Proc. ACM ASIACCS 2016. pp. 141–151
37. Zimmermann, V.: From the quest to replace passwords towards supporting secure and usable password creation. Ph.D. thesis, Technische Universität of Darmstadt

A Discussion of vault capacity

During the encryption process of the honey vault scheme, the plaintext space $\mathcal{P}$ is completely determined by the selected set of passwords and DTE, the ciphertext space $\mathcal{C}$ is full space, and the key space $\mathcal{K}$ is the set of all possible master passwords. Correspondingly, there are random variables $\mathbf{P}$, $\mathbf{C}$ and $\mathbf{K}$. For vaults that contain n passwords, we denote the distribution by $\mathbf{P}^n$. Suppose V is the language composed of all the vaults, we give an asymptotical definition of the entropy of V as follows:

$$H_V = \lim_{n\to\infty} \frac{H(\mathbf{P}^n)}{n}$$

and the redundancy of V is defined to be

$$R_V = 1 - \frac{H_V}{\log_2 |\mathcal{P}|}.$$

Given probability distributions on $\mathcal{K}$ and $\mathcal{P}^n$, we can induce the probability distribution on $\mathcal{C}^n$, and define $\mathbf{C}^n$ to be a random variable of a ciphertext representing n passwords. For the stored ciphertext y$\in \mathcal{C}^n$,define

$$K(\mathbf{y}) = \{K \in \mathcal{K} : \exists \mathbf{x} \in \mathcal{P}^n \text{ such that } \Pr[\mathbf{x}] > 0 \text{ and } e_K(\mathbf{x}) = \mathbf{y}\}.$$

as the set of plausible-looking keys. Then the number of decoy keys is $|K(\mathbf{y})|-1$, because only one of these plausible-looking keys is the correct key. The average number of decoy keys (over all possible ciphertext corresponding to n passwords) is denoted by $\bar{s}_n$. Its value is computed to be

$$\begin{aligned}\bar{s}_n &= \sum_{\mathbf{y}\in\mathcal{C}^n} \mathbf{Pr}[\mathbf{y}](|K(\mathbf{y})| - 1) \\ &= \sum_{\mathbf{y}\in\mathcal{C}^n} \mathbf{Pr}[\mathbf{y}]|K(\mathbf{y})| - \sum_{\mathbf{y}\in\mathcal{C}^n} \mathbf{Pr}[\mathbf{y}] \\ &= \sum_{\mathbf{y}\in\mathcal{C}^n} \mathbf{Pr}[\mathbf{y}]|K(\mathbf{y})| - 1.\end{aligned}$$

We can relate the $H(\mathbf{K}|\mathbf{C}^n)$ to $\bar{s}_n$ as follow:

$$\begin{aligned}H(\mathbf{K}|\mathbf{C}^n) &= \sum_{\mathbf{y}\in\mathcal{C}^n} \mathbf{Pr}[\mathbf{y}]H(\mathbf{K}|\mathbf{y}) \\ &\le \sum_{\mathbf{y}\in\mathcal{C}^n} \mathbf{Pr}[\mathbf{y}] \log_2 |K(\mathbf{y})| \\ &\le \log_2 \sum_{\mathbf{y}\in\mathcal{C}^n} \mathbf{Pr}[\mathbf{y}]|K(\mathbf{y})| \\ &= \log_2(\bar{s}_n + 1).\end{aligned}$$

According to the property of the cryptosystem [29], we have that

$$H(\mathbf{K}|\mathbf{C}^n) = H(\mathbf{K}) + H(\mathbf{P}^n) - H(\mathbf{C}^n).$$

For a sufficient large n, we can use the estimate

$$H(\mathbf{P}^n) \approx nH_{\mathrm{V}} = n(1 - R_V) \log_2 |\mathcal{P}|.$$

Note the fundamental property of entropy,

$$H(\mathbf{C}^n) \le n \log_2 |\mathcal{C}|.$$

Then, if $|\mathcal{P}| = |\mathcal{C}|$, it follows that

$$H(\mathbf{K}|\mathbf{C}^n) \ge H(\mathbf{K}) - nR_V \log_2 |\mathcal{P}|.$$

Combining the two inequalities about $H(\mathbf{K}|\mathbf{C}^n)$, we get that

$$\log_2(\bar{s}_n + 1) \ge H(\mathbf{K}) - nR_V \log_2 |\mathcal{P}|.$$

It can be intuitively seen that, for a sufficient large n, the inequality would degenerate into trivial form $\bar{s}_n \ge 0$, which implies that *the existence of the decoy vault is not guaranteed.*