How to Design Secure Honey Vault Schemes

Zhenduo Hou
Nankai University
Tianjin, China
joehou13@nankai.edu.cn

Tingwei Fan
Nankai University
Tianjin, China
fantingwei@mail.nankai.edu.cn

Fei Duan
Nankai University
Tianjin, China
duanf@mail.nankai.edu.cn

Ding Wang
Nankai University
Tianjin, China
wangding@nankai.edu.cn

Abstract

Password vaults enable a user to store multiple passwords with a single master password. Honey encryption (HE) protected password vaults (called honey vaults), are promising in resisting offline master password guessing attacks. Trial-decrypted with incorrect master passwords, honey vaults are designed to yield plausible-looking decoy vaults to confuse attackers, forcing them to perform online verifications to know whether a decrypted vault is the real one.

In this paper, we demonstrate how to design secure honey vault schemes in a *principled* approach. We first identify three major types of vulnerabilities, and propose three critical design criteria based on rigorous theories, with each aiming to address one type of vulnerability. These criteria are: (1) Employing an accurate password probability model (PPM) in the natural language encoder (NLE, a key component of a honey vault) to resist distribution-aware distinguishing attacks; (2) Employing sequence-based PPMs for unique passwords, and sufficiently concise reuse models to resist encoding attacks (USENIX SEC'19); (3) Hiding a user's real-vault-related (i.e., adaptive) PPM to resist extraction attacks (USENIX SEC'21). To meet these key criteria, we propose VaultGuard with an innovative NLE and HE-Adaptive to honey-encrypt a user's real vault and the adaptive PPM, respectively. Our NLE eliminates the first and second vulnerabilities, while HE-Adaptive addresses the third. Security evaluations on real-world data reveal that our VaultGuard can significantly enhance honey vault security, forcing attackers to perform 1.10~3.98 times online verifications. We also provide an efficient proof-of-concept VaultGuard implementation on the client side. We believe this work provides general principles and actionable guidelines for designing secure honey vault schemes.

CCS Concepts

• **Security and privacy → Authentication**.

Keywords

Password Managers; Honey Vaults; Distinguishing Attacks

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '25, Taipei.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765162

ACM Reference Format:
Zhenduo Hou, Tingwei Fan, Fei Duan, and Ding Wang. 2025. How to Design Secure Honey Vault Schemes. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3719027.3765162

1 Introduction

Passwords stubbornly survive as the main authentication method. Although they suffer from issues in both security and usability (e.g., guessing [40, 70, 72], phishing [48, 57], and memorization [27]), and various alternatives (e.g., multi-factor [28, 30] and FIDO2 [4]) have been proposed, passwords survive due to their remarkable simplicity, easiness to change and low deployment costs [13, 14, 80].

As the number of accounts possessed by users continues to increase, human memorability becomes one of the primary constraints on password usability. For instance, a recent survey of 1,509 users reveals that an ordinary user has 168 different accounts on average [59]. Since it is unlikely to memorize a different password for each account, users often resort to reusing passwords, making them vulnerable to various reuse-based guessing attacks (see [46, 67, 69, 75]). To reduce the burden of memorizing multiple passwords, password vaults (i.e., password managers) are widely recommended by both academia and industry [21, 31, 43, 53, 62, 71].

In general, a user's daily-used passwords are encrypted by a selected master password, and stored in the vault along with site domains and usernames. Hence, *a user only needs to memorize one master password instead of all possessed passwords.* However, such a password vault is a rich target for offline guessing attackers: Once the password vault is leaked, the attacker $\mathcal{A}$ can offline guess the user's master password and trial-decrypts the encrypted vault. When $\mathcal{A}$ tries an incorrect master password, the vault will be decrypted as a non-ASCII string, signaling its incorrectness (see Fig. 1). Moreover, $\mathcal{A}$ can leverage existing password cracking software (e.g., Hashcat [54] and JtR [49]) and hardware (e.g., RTX 5090 [23]) to facilitate offline attacks to perform 10^9~10^{12} guesses per day, posing a major threat against conventional password vault schemes.

Worse still, these days people are increasingly familiar with catastrophic password vault data breaches. For instance, the browser-based Opera password manager leaked vaults of 1.7 million users in 2016 [45], and the leading password manager LastPass leaked millions of password vaults *three times* in 2011, 2015, and 2022 [52, 58, 74]. Particularly, the 2022 LastPass breach resulted in $150 million cryptocurrency asset theft in 2024, where attackers recovered leaked LastPass vaults by offline guessing master passwords,

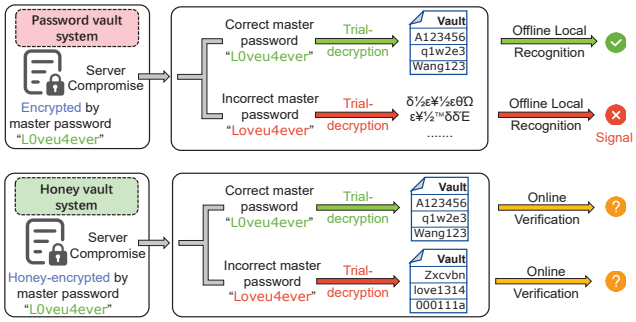


Figure 1: An illustration of the differences between conventional password vault schemes (the upper part) and honey vault schemes (the lower part). Once the server is compromised, the attacker can perform offline guessing attacks. The decrypted non-ASCII random strings will signal attackers. While in a honey vault scheme (the lower part), an incorrect trial-decryption can still yield plausible-looking decoy vaults to confuse the attacker, and online verifications are further required (which can be better detected and mitigated).

and successfully extracted crypto-wallet passwords [3, 51]. Such security events also severely impact users' adoption of password managers. A recent study has found that 67% (36/54) of the users investigated lost trust in LastPass, and all six users who stopped using LastPass after the 2022 data breach reported doing so due to security concerns [42]. Thus, it is important to make password vaults resistant to offline master password guessing attacks.

To address this issue, defense-in-depth security mechanisms on both the server and the client sides are recommended. However, on the server side, critical drawbacks impede the deployment of representative password-hardened encryption (PHE) [37]. First, PHE requires an external crypto-server [37], which can be expensive in practice. Second, when the crypto-server fails, a single point of failure prohibiting normal authentication occurs, and addressing it requires even more crypto-servers (e.g., 3∼15) [15]. Third, once PHE is compromised, the overall security can be reduced: $\mathcal{A}$ can obtain sensitive information (e.g., users' login patterns) that is essential for the crypto-server to rate-limit online login attempts [37], so she may better circumvent defenses against online master password guessing attacks [34]. Thus, we do not consider PHE in this work.

On the client side, honey encryption (HE) protected password vaults, called honey vaults, have been proposed. The security goal of a honey vault scheme is to generate decoy vaults that are sufficiently indistinguishable from a user's real vault. As Fig. 1 shows, a honey vault can produce plausible-looking decoy vaults rather than non-ASCII strings when $\mathcal{A}$ tries incorrect master passwords, forcing $\mathcal{A}$ to use online verifications on websites to find if a decrypted vault is real. This not only reduces $\mathcal{A}$'s guess number from billions [23] to tens of hundreds per day (e.g., 120∼1,440 consecutive failed logins per day [65]), but also makes $\mathcal{A}$ more likely to be detected by online defenses (e.g., rate-limiting, see [9, 33, 34, 39, 65]) deployed on websites. Unlike PHE, a honey vault scheme: (1) is easy to deploy and avoids the single point of failure, as no external crypto-server is needed [17]; (2) is at least as secure as the conventional vault when HE is ineffective; (3) and records no data about users' master passwords [17, 19, 29], so $\mathcal{A}$ can obtain no extra guessing advantages. All this makes the honey vault technique a promising solution.

A honey vault scheme achieves the above goal by utilizing (1) the natural language encoder (NLE), which comprises a password probability model (PPM) and a distribution transforming encoder (DTE), and (2) the conventional password-based encryption (PBE, e.g., AES with PBKDF2, see [17, 19, 29, 36]) that uses a user's master password as the key. The PPM comprises a unique password probability model (i.e., UPPM like PCFG [72] and Markov [40]) for unique passwords, and possibly, a reuse model (RPPM, see [19, 29]) for reused ones. At a high level, a honey vault scheme works as follows. First, it parses each password in a user's real vault into a vector of generation rules through the PPM. Then, each rule vector is encoded into a bit string (i.e., code) via DTE and is then encrypted via PBE (see more in Secs. 2.1 and 2.2). The decryption is the opposite. In this way, when $\mathcal{A}$ tries incorrect master passwords, different codes will be decrypted and further decoded into decoy vaults following the same distribution determined by the PPM.

1.1 Motivations and challenges

Conventional password vault schemes cannot resist offline master password guessing attacks. The server-side PHE mitigation can be expensive, and cause a single point of failure, and even reduce the overall security. In comparison, a honey vault deployed on the client side can address these flaws, making it a promising solution. In this work, we focus on human-generated passwords because (1) users tend to create passwords on their own even when password managers are available [44, 79]: A recent survey shows that only 27% of the investigated 1,012 password manager users adopt randomly generated passwords due to trust concerns [60]; (2) Chatterjee et al.'s UNIF [17] can readily generate indistinguishable decoys for randomly generated passwords. In contrast, for human-generated passwords, existing honey vault schemes [17, 19, 20, 29, 50] fall short of generating sufficiently indistinguishable decoy vaults. Three major challenges exist, and we explain them as follows.

Firstly, a secure honey vault scheme should defend against various distribution-aware distinguishing attacks that exploit differences between the real and decoy vault distributions. This category comprises multiple distinguishing attacks (e.g., Golla et al.'s KL divergence attack [29] and Duan et al.'s theoretically-grounded attack [25]), so it is crucial to find the most effective attack approach, and design a honey vault scheme that can resist it. The designed scheme is expected to resist not only the most effective, but also other distribution-aware attacks. This task is challenging, as it requires an in-depth understanding of the inherent relationships between various honey vault distinguishing attacks.

Secondly, it is unclear PPMs of what characteristics can be used to build secure honey vault schemes. In 2019, Cheng et al. [20] found ambiguities in the PCFG-based NoCrack-NLE [17]. For example, "`password`" can be seen as a single word and a combination of words "`pass`" and "`word`" [20], but only one segmentation can be used in encoding the real vault. Thus, other segmentations of the same content are decoys. By exploiting such ambiguities, they proposed weak and strong encoding attacks [20]. As a potential mitigation, they proposed IS-PMTE that samples generation rules based on their probabilities in parsing ambiguous password segments [20]. However, whether it is secure against distribution-aware attacks has not been evaluated, and the exact time complexity has not been rigorously quantified. Addressing this challenge requires detailed investigations of the sophisticated IS-PMTE workflow [20].

Thirdly, inconsistent criteria need to be simultaneously satisfied. Decoy vaults independent of the real vault are vulnerable to

distribution-aware attacks, but making them similarly distributed can facilitate other distinguishing attacks. For instance, to resist the KL-divergence attack [29], Golla et al.'s adaptive mechanism [29] boosts (e.g., multiplies the frequency by 5) one 4-gram in each password in the real vault, and other 4-grams with a probability of 20%. However, following the Kerckhoffs's principle, attackers can know which 4-grams are boosted, and use Cheng et al.'s adaptive extra (meaning extraction) attack [19] whose one-guess success rate can be 45%. Resisting such attacks is challenging, as effective defenses should: (1) not reduce the NLE security against distribution-aware attacks; (2) make decoy vaults and adaptive PPMs maintain the same relation as the real one (e.g., each password has one 4-gram in the adaptive PPM [29]) to resist extraction attacks that exploit inconsistencies between decoy vaults and adaptive PPMs.

1.2 Related work

At IEEE S&P'15, Chatterjee et al. [17] first identified security flaws in Bojinov et al.'s Kamouflage password vault that explicitly stores the real vault with a number (e.g., 10^7) of decoy vaults [12]. As a countermeasure, they proposed the framework of the honey vault scheme, which comprises a natural language encoder (NLE) to encode passwords into random bit strings (i.e., codes), and conventional password-based encryption (e.g., AES with PBKDF2) that encrypts codes into ciphertexts. In decryption, a *single* ciphertext can be decoded into various plausible-looking decoy vaults when $\mathcal{A}$ tries multiple incorrect master passwords [17]. As a concrete instantiation, they proposed NoCrack-NLE that employs PCFG UPPM [72] and their designed Chatterjee-DTE as the first honey vault scheme [17]. They also provided, to the best of our knowledge, the only publicly available password vault dataset `Pastebin` [17].

At CCS'16, Golla et al. [29] claimed that NoCrack-NLE [17] cannot resist distribution-aware distinguishing attacks, and proposed the Kullback–Leibler (KL) divergence attack by exploiting the difference between password distributions in real and generated decoy vaults [29]. On average, their attack can identify the real vault within the first 6.20% (not the ideal 50% in random guesses) online guesses/verifications [29]. To make decoy vaults more indistinguishable, they proposed an adaptive mechanism [29] that deterministically boosts (e.g., multiplies the frequency by 5) selected 4-grams in the real vault, and other 4-grams in a UPPM with a probability of 20% (see Sec. 2.2). They also devised a heuristic reused password probability model (RPPM) to capture users' reuse behaviors.

At USENIX SEC'19, Cheng et al. [20] exploited ambiguities in rules (in the form of a vector) of PCFG-based NoCrack-NLE [17] and the RPPM in Golla-NLE [29], and proposed the weak and strong encoding distinguishing attacks [20]. For example, in PCFG [72], "`password`" can be seen as a single word, and the combination of "`pass`" and "`word`", but only one rule (e.g., with the highest probability) is used to encode the real vault. Hence, if a decrypted code is decoded into other rule vectors, its vault must be a decoy. As a defense, Cheng et al. [20] proposed IS-PMTE that randomly selects one rule vector over *all* ambiguous rules in a vault. However, as we will show in Sec. 3.3, their solution is vulnerable to distribution-aware attacks, and suffers from exponential time complexity.

At USENIX SEC'21, Cheng et al. [19] explored the scenario where an attacker has obtained both an old (e.g., backup) and a new version of a honey vault. As a defense, they proposed an incremental update (IU) mechanism to ensure that, as the real vault, decrypted codes of old and new decoy vaults share the same prefix. They also proposed a new RPPM to better mitigate ambiguities in Golla-NLE [29]. They designed a number of practical (e.g., single password and password similarity) attacks based on their real-to-decoy ratios [19]. In particular, their adaptive extra and hybrid attacks [19] against Golla et al.'s adaptive mechanism [29] exploit the inherent relationship between the real vault (which does not exist for decoy vaults) and the adaptive PPM correlated with it, and identify the real vault within the first 7%∼29% online verifications. As mitigation, they proposed using 6-gram Markov UPPM [19]. However, as we will show in Sec. 4.1, their scheme suffers from heavy storage overhead.

At IEEE S&P'24, Duan et al. [25] calculated the Bayesian posterior probability of vaults when given ciphertexts, and proposed the theoretically-grounded attack. They also proposed the super encoding attack that exploits the difference between codes encoded from Chatterjee-DTE [17] (used in NoCrack-NLE [17] and Golla-NLE [29]) and uniformly distributed codes. Attack instantiations with real-world PPMs (e.g., $\frac{1}{3}$List+$\frac{1}{3}$PCFG+$\frac{1}{3}$Markov [25] with TarGuess-II [67] or Pass2Path [46]) reveal that their attack is 1.15∼4.35 times as efficient as others. At ESORICS'23, Rao et al. [50] proposed a master password guessing attack (MPGA) that exploits users' master password resets, and reaches a near 100% success rate.

1.3 Our contributions

- **Three design criteria**. We identify three major types of vulnerabilities in existing honey vault (HV) schemes and accordingly propose design criteria. Specifically, they are: (1) PPMs should accurately model users' password distributions for both unique and reused passwords in the vault; (2) Sequence-based UPPMs should be employed and RPPM should be sufficiently concise to avoid various encoding attacks; (3) Users' adaptive PPMs should be inaccessible to attackers trying incorrect master passwords to resist adaptive extract/hybrid attacks [19]. For each criterion, we develop both rigorous theories and extensive experiments to show their necessity. Overall, these criteria serve as general guidelines for designing practical and secure HV schemes.
- **An innovative VaultGuard solution**. Based on our design criteria, we explore state-of-the-art UPPMs and RPPMs for unique and reused passwords, and propose our VaultGuard-NLE with an innovative honey-encrypted adaptive mechanism named HE-Adaptive as the solution. First, VaultGuard-NLE leverages the accurate 5-gram Markov sequence-based UPPM and our proposed RPPM, and it is ambiguity-free to resist encoding attacks [20]. Second, our proposed HE-Adaptive treats the user's adaptive PPM as the secret, and makes it inaccessible to attackers trying incorrect master passwords. Our experiments show that HE-Adaptive can mitigate adaptive extra attacks without bringing new risks.
- **Extensive evaluations**. Evaluation results on real-world data reveal that our VaultGuard (with HE-Adaptive) can significantly enhance the security against various honey vault distinguishing attacks, forcing attackers to perform 1.10∼ 3.98 times online verifications to identify the real vault. In particular, HE-Adaptive alone forces an attacker to perform 1.02∼1.71 times online verifications, and we provide

a proof-of-concept implementation of VaultGuard as a browser-based extension. Performance evaluations show that the latency is sub-second, so VaultGuard is efficient.
- **Some discussions**. For security, we discuss the impacts of our VaultGuard improvements in real-world attack scenarios, and the ways that can mitigate threats posed by websites allowing large or even unlimited online guesses. For usability, we discuss potential ways that can help users identify and recover from undesirable master password typos, as well as their security and usability/deployability trade-offs.

2 Background

Notation. The notation $\coloneqq$ denotes the deterministic assignment, and $\leftarrow$ denotes the random assignment (sampling) based on a distribution. Let $\mathcal{S}$ be a finite set and $|\mathcal{S}|$ be its size. A distribution on set $\mathcal{S}$ is a function $\Pr_S(\cdot) : \mathcal{S} \rightarrow [0,1]$, such that $\sum_{s\in S} \Pr_S(s)=1$. The notation $s \xleftarrow{\$} \mathcal{S}$ denotes sampling an element $s \in \mathcal{S}$ uniformly at random, and $[a, b)$ denotes the set $\{a, ..., b-1\}$, where $a, b \in \mathbb{Z}$.

2.1 Honey encryption

Honey encryption (HE) is a *symmetric encryption application* suited for low-entropy secrets (e.g., passwords) [36]. A HE scheme contains a distribution transformation encoder (DTE) and a conventional symmetric password-based encryption scheme (PBE). The DTE is a public pair comprising a randomized encoder and a deterministic decoder. In encryption, the DTE encodes a message $m \in \mathcal{M}$ into a fixed-length code $s \in \mathcal{S}$, then the PBE encrypts s into a ciphertext $c \in C$. The decryption is the opposite. In this framework, a trial-decryption on the ciphertext c with an incorrect key (e.g., from the offline guessing attacker $\mathcal{A}$) yields a different code $s^* \in \mathcal{S}$, which is then decoded into a different message $m^* \in \mathcal{M}$ instead of non-ASCII random strings. In this way, HE confuses $\mathcal{A}$ from knowing if her tried keys are correct to resist offline guessing attacks [35, 36].

Instantiations of DTE. At EUROCRYPT'14, Juels and Ristenpart proposed the first practical DTE, i.e., IS-DTE, using the inverse sampling technique [36]. The IS-DTE encodes a message into a code of length l and conversely, decodes any code of length l into a message. Suppose messages are ranked in a given (e.g., lexicographical) order in the form $\{m_1, \ldots, m_{|\mathcal{M}|}\}$, and $\Pr_{\mathcal{M}}(\cdot)$ denotes the probability distribution on $\mathcal{M}$. Let $\mathsf{CDF}(\cdot)$ be the cumulative distribution function with $\mathsf{CDF}(m_0) \coloneqq 0$ and $\mathsf{CDF}(m_i) \coloneqq \sum_{t=0}^{i} \Pr_{\mathcal{M}}(m_t)$. Upon these premises, the code length l and factor ρ satisfy $\rho \coloneqq \frac{1}{2^l} \ll \min_m \Pr_{\mathcal{M}}(m)$, and the representation function $\mathsf{rp}_\rho(\cdot)$ specified by ρ is $\mathsf{rp}_\rho(a) \coloneqq \arg\min_{\gamma\in\mathbb{N}} |a - \gamma\cdot\rho|$ for any value $a \in [0,1]$. In encoding, the code s of the message $m_i \in \mathcal{M}$ is uniformly sampled from m_i's cumulative distribution interval in the way $s \xleftarrow{\$} [\mathsf{rp}_\rho(\mathsf{CDF}(m_{i-1})), \mathsf{rp}_\rho(\mathsf{CDF}(m_i)))$. To decode the code s, one can just determine the interval where s is located and then get the corresponding m_i.

Besides IS-DTE, Chatterjee et al. have also proposed a DTE [17] to encode fractions of the form $\frac{p}{q}$, where $p, q \in \mathbb{N}$ and $p \le q$. In this paper, we mainly use IS-DTE since Chatterjee's DTE is vulnerable to Duan et al.'s super encoding distinguishing attack [25].

2.2 Honey vault schemes

In general, a honey vault (HV) is a password vault protected by honey encryption: It treats a user's master password as the encryption key and passwords stored in it as the messages. From a user's view, a honey vault is the same as a conventional vault except that incorrect master passwords (e.g., typos, which may occur at a rate of 4.65%~8.06% [16, 18]) lead to plausible-looking vaults rather than explicit login failure signals (e.g., notifications and non-ASCII strings, see Fig. 1). We present the key components as follows.

Natural language encoder (NLE) is the core of a honey vault scheme that encodes a vault into a code and decodes an arbitrary code into a vault. An NLE contains two sub-modules: a password probability model (PPM, e.g., PCFG [72] and Markov [40] with a reuse model like the ones in [19, 29]) and a DTE. When encoding a vault, the PPM parses each password in the vault into a rule vector, and assigns each rule with its probability given by the PPM. Then the DTE encodes these rules into codes. The decoding is the opposite. Since existing reuse models [19, 29] are mostly straightforward in modeling password reuses, we mainly focus on UPPM, and divide them into segment-based and sequence-based categories.

Segment-based UPPM parses a password into disjoint segments based on its grammars, and fills content for each segment. Two major segment-based UPPMs, i.e., PCFG [72] and PwdSegment [76], exist. Suppose a password pw can be parsed into the rule vector $\mathbf{rv}=(\mathsf{S}\rightarrow\mathcal{T}_{pw}=\prod_{m\in[1,M]} \mathsf{T}^{(m)}_{l_{m,i,j}}, \mathsf{T}^{(m)}_{l_{m,i,j}} \rightarrow \mathtt{str})$, where m identifies the type of the template, l_m denotes its length, i identifies the position of pw in the vault, j identifies the position of the parsed segment in pw, and `str` denotes the string content of the template $\mathsf{T}^{(m)}_{l_m,i,j}$, respectively. For better illustration, we use "`p@ssw0rd4ever`" as an example to show how PCFG [72] and PwdSegment [76] work.

PCFG comprises three different types of templates: letter L, digit D, and symbol S [72]. Thus, the password is divided into $\mathsf{L}_1\mathsf{S}_1\mathsf{L}_3\mathsf{D}_1\mathsf{L}_2\mathsf{D}_1\mathsf{L}_4$, and its probability is calculated as:

$$\begin{aligned}\Pr_{\mathrm{PW}}(\mathtt{p@ssw0rd4ever}) = \Pr_{\mathrm{PW}}(\mathsf{L}_1\mathsf{S}_1\mathsf{L}_3\mathsf{D}_1\mathsf{L}_2\mathsf{D}_1\mathsf{L}_4)\cdot\Pr_{\mathrm{PW}}(\mathtt{p}|\mathsf{L}_1)\cdot\Pr_{\mathrm{PW}}(\mathtt{@}|\mathsf{S}_1)\\ \cdot\Pr_{\mathrm{PW}}(\mathtt{ssw}|\mathsf{L}_3)\cdot\Pr_{\mathrm{PW}}(\mathtt{0}|\mathsf{D}_1)\cdot\Pr_{\mathrm{PW}}(\mathtt{rd}|\mathsf{L}_2)\cdot\Pr_{\mathrm{PW}}(\mathtt{4}|\mathsf{D}_1)\cdot\Pr_{\mathrm{PW}}(\mathtt{ever}|\mathsf{L}_4),\end{aligned} \tag{1}$$

where $\Pr_{\mathrm{PW}}(\cdot)$ denotes the probability of a password in the PPM.

PwdSegment uses Byte-Pair Encoding algorithm [26] to merge as frequent characters and segments as possible [76]. Besides L, D, and S, PwdSegment [76] comprises two additional types of templates: DM (i.e., double mixed) composed of two types of characters, and TM (i.e., triple mixed) composed of three types of characters. For simplicity, we do *not* discriminate between upper and lower case letters as in [76]. Therefore, in PwdSegment, "`p@ssw0rd4ever`" is divided into $\mathsf{TM}_8\mathsf{DM}_5$, and its probability is calculated as:

$$\begin{aligned}\Pr_{\mathrm{PW}}(\mathtt{p@ssw0rd4ever}) =& \Pr_{\mathrm{PW}}(\mathsf{TM}_8\mathsf{DM}_5)\cdot\Pr(\mathtt{p@ssw0rd}|\mathsf{TM}_8)\cdot\\ &\Pr_{\mathrm{PW}}(\mathtt{4ever}|\mathsf{DM}_5).\end{aligned} \tag{2}$$

Sequence-based UPPM treats a password $pw=c_1\ldots c_l$ as a d-gram sequence, and assumes that pw is created from the leftmost to the rightmost. The generation rules are of the form $\mathbf{rv}=((c_i|c_{i-d+1}\cdots c_{i-1}))_{i=1}^{l}$, and typical d-gram UPPMs are Markov [40], FLA [41], and RFGuess [70]. In this framework, regardless of how password probabilities are generated, the probability of a rule vector is:

$$\Pr_{\mathrm{PW}}(c_1c_2\ldots c_l) = \prod_{i=1}^{d}\Pr_{\mathrm{PW}}(c_i \mid c_{i-d+1}\ldots c_{i-1}), \tag{3}$$

where $\Pr_{\mathrm{PW}}(c_i|c_{i-d+1}\ldots c_{i-1}) \coloneqq \frac{\mathsf{Count}(c_{i-d+1}\ldots c_{i-1}c_i)}{\mathsf{Count}(c_{i-d+1}\ldots c_{i-1})}$, and $\mathsf{Count}(\cdot)$ is the frequency of a string in the trained model. For instance, the rule vector of all 4-grams of "`qwerty`" is $\mathbf{rv}$=(`q|***`, ..., `r|qwe`,

t|wer, y|ert, ⊥ |rty), where * and ⊥ are the start and end symbols, respectively. To overcome the sparsity issue, end-symbol and distribution-based smoothings are also incorporated [40].

Golla et al.'s adaptive mechanism [29] adjusts distributions in decoy vaults based on the user's real vault. By boosting d-gram probabilities, the sequence-based Markov UPPM [40] does not give very low probabilities for d-grams in the real vault. More concretely, their probabilities are adjusted as follows: (1) For each password in the real vault, the probability of a randomly chosen d-gram is multiplied by 5; (2) For the rest of d-grams, they have a 20% chance to be increased by 5 times; (3) All d-grams are renormalized.

2.3 Security model and evaluation metrics

Honey vault (HV) distinguishing attack. The core security goal of HV schemes is to make generated decoy vaults indistinguishable. This goal is defined against an HV distinguishing attacker $\mathcal{A}$ who has obtained the encrypted vault (e.g., from breaches) and can try (almost) all possible master password guesses to trial-decrypt the ciphertext. These attacks are summarized in Table 1 (see more details in Appendix A). Following the Kerckhoffs's principle, we assume that $\mathcal{A}$ does not know any prior information about passwords in the vault, but the exact NLE and PBE. In evaluation, we assume that $\mathcal{A}$ has obtained n=1,000 (the exact value depends on the guessability of a user's master password) plausible-looking vaults as in prior studies [17, 19, 20, 25, 29, 50], with the real vault among them. Particularly, the attack process goes as follows:

- $\mathcal{A}$ uses n candidate master passwords to trial-decrypt the ciphertext, and gets n plausible-looking vaults;
- $\mathcal{A}$ assigns a score to each of the n vaults via a score function $\mathsf{s}(\cdot)$ determined by the employed (e.g., KL divergence [29]) distinguishing attack approach, and then online logs in the descending order to verify whether the vault is real.

Other attacks, such as phishing [48, 57] and memory extractions (e.g., CVE-2023-32784 [38]), obtain a user's real vault in non-cryptographic ways, so we do not consider them in this work.

Average rank $\bar{r}$ and accuracy α security metrics measure the *average-case* performance of an HV scheme against an HV distinguishing attack. In short, for each HV scheme and attack, $\bar{r}$ is the average fraction (i.e., $0 \le \bar{r} \le 1$) of online guesses/verifications $\mathcal{A}$ needs to identify the real vault among n vaults, and α is the average accuracy of distinguishing the real vault from decoys [17]. *Since there exists $\alpha = 1-\bar{r}$ [20], we mainly use the average rank metric in this work.* Formally, we summarize the security definition as follows.

Definition 1. *The average rank $\bar{r}$ of an HV scheme (depending on its NLE) against an HV distinguishing attack has $\bar{r} = \frac{\bar{g}-1}{n-1}$ [20], where $\bar{g}$ is the absolute number (on average) of $\mathcal{A}$'s online verifications to identify the real vault v among the n decrypted vaults. An ideally secure HV scheme should have $\bar{r}$=0.5 against all HV distinguishing attacks, and the closer $\bar{r}$ is to 0.5, the more secure an HV scheme is.*

By definition, an ideally secure HV scheme can resist $\mathcal{A}$'s HV distinguishing attacks by forcing $\mathcal{A}$ to online guess/verify randomly: No matter how sophisticated distinguishing attacks are, $\mathcal{A}$ cannot gain extra advantages over guessing uniformly at random (where $\bar{r}$=0.5), and more uniform guesses mean a more secure HV scheme. Similarly, an ideally effective attack should have $\bar{r}$=0 and α=1, i.e., the real vault can be identified with only *one* guess.

This definition is practical because: (1) It essentially captures the capability that an HV scheme can disguise/hide the real vault without being impacted by user-specific master passwords; (2) The average rank remains quite stable across varied n [17, 29], and can be translated into more intuitive security metrics when n is known/estimated (e.g., by password strength meters [65]). Particularly, the absolute number of online verifications can be calculated as $\bar{g} = \bar{r}(n-1)+1 \approx \bar{r} \cdot n$ due to $n \gg \bar{r}$, which enables security administrators to take appropriate mitigations (see details in Sec. 6.1).

Cumulative distribution graph (CDG) [20] metric extends the average rank, and plots the average fraction y of identified vaults with $\mathcal{A}$'s x online verifications (measured in fractions), exhibiting the global attack efficiency. The closer a curve is to the y=x baseline, the more secure the HV scheme is against a distinguishing attack.

2.4 Datasets and ethical considerations

Datasets. Our evaluation relies on Pastebin, which, to the best of our knowledge, is the *only* public password vault dataset. Pastebin may have been leaked in 2011 and was made public in 2016 [17]. It contains 276 vaults, and stores 2∼50 username-password pairs in each vault. To better simulate the asymmetric knowledge between attackers and HV schemes in the real world, we choose the Wishbone dataset leaked in 2020 to train HV attackers, and the Rockyou dataset leaked in 2009 to train NLEs, respectively. The 10 million Wishbone provides sufficient data for attackers to learn users' password creation behaviors and is used in recent studies [25, 70], while the 33 million Rockyou is one of the most widely used datasets in password studies (e.g., [17, 19, 20, 29, 40, 41, 46]).

Ethical considerations. Although these datasets are publicly available on the Internet and have been widely used in prior studies [17, 19, 20, 25, 29], we process them with caution. More specifically, (1) we do not further disseminate leaked datasets, and only use them for this study; (2) Our datasets are stored and processed on computers unconnected to the Internet; (3) We treat each account confidential, and mainly report aggregated statistics to minimize the impacts to affected users; (4) We delete all sensitive information after our analysis. Through these precautions, our work meets the key ethical guidelines [56]. We have also obtained approval from our center's IRB. Overall, our work facilitates a better understanding of honey vault security, thereby benefiting the community.

3 Design Criteria for Honey Vaults

3.1 Overall design idea

To generate as indistinguishable decoy vaults as possible (i.e., the shield), one must understand HV distinguishing attacks (i.e., the sword). Existing attacks mainly exploit three types of vulnerabilities: (1) differences between password distributions in real and decoy vaults (e.g., KL divergence [29] and Duan et al.'s theoretically-grounded attacks [25]); (2) ambiguous password generation rules (e.g., encoding attacks [20]); (3) the publicity of the user's adaptive PPM that is accessible to $\mathcal{A}$ (e.g., adaptive extra attack [19]).

We propose three design criteria (see Table 1) for the above vulnerabilities. First, we explore distribution-aware distinguishing attacks. By employing the Stackelberg (i.e., the leader-follower) framework [61] in game theory, we *prove two key theorems*: (1) An NLE can reduce the theoretically-grounded distinguishing attack [25] to guesses uniformly at random if and only if the real and decoy

vaults are uniformly distributed; Otherwise, the best an NLE can do is forcing $\mathcal{A}$ to guess in the descending order of the real-world vault probability (see Theorem 1); (2) An NLE resisting the theoretically-grounded attack [25] can also resist other distribution-aware attacks like KL divergence [29], single password and password similarity attacks [19] (see Theorem 2). Hence, *only the theoretically-grounded attack [25]* is needed in evaluating HV security, and we propose Criterion #1 for designing secure HV schemes. As a byproduct, we also evaluate the security impacts of reuse models (i.e., RPPMs).

Second, we reveal that ambiguous rules are rooted in NLEs with segment-based UPPMs (see Sec. 2.2), and the only feasible Cheng et al.'s IS-PMTE-style solution [20] is *inherently* vulnerable to distribution-aware attacks. We also reveal that this solution suffers from exponential time complexity even if its workflow has been intensively improved (see Sec. 3.3). Hence, such NLEs should be avoided from scratch (Criterion #2). Third, we demonstrate that it is *inherently impossible to maintain security* if the PPM correlated with the user's real vault (e.g., via Golla et al.'s adaptive mechanism [29]) is exposed to attackers (which is intrinsic following the Kerckhoffs's principle). To resist such attacks, the user's adaptive PPM should be hidden/masked from attackers (Criterion #3, see Sec. 3.4).

3.2 Analysis on Criterion #1

We prove the two key theorems in this section. To do this, we first recall the theoretically-grounded attack [25], and provide the formal definition for NLE resisting it and other distinguishing attacks. We denote $\Pr_{\mathrm{PW}}(\cdot)$, $\Pr_{\mathrm{V}}(\cdot)$, $\Pr_{\mathrm{MP}}(\cdot)$, $\Pr_{\mathrm{C}}(\cdot)$, $\Pr_{\mathrm{D}}(\cdot)$ as the probabilities of real passwords (usually unknown, but can be approximated), real vaults (also unknown), master passwords, ciphertexts, and decoy vaults sampled from the NLE, respectively. The calligraphic notations (e.g., $\mathcal{PW}$, $\mathcal{V}$ and $\mathcal{MP}$) denote the corresponding spaces.

Theoretically-grounded attack [25] obtains the Bayesian posterior probability by enumerating all possible master passwords and codes, and uses it as the score function. Suppose the user's master password is mp, the real vault is v, and all encoded codes s of v are uniformly distributed over $\mathcal{S}$ when IS-DTE is applied [36]. When a user randomly selects a master password $mp \in \mathcal{MP}$ from $\Pr_{\mathrm{MP}}(\cdot)$ and encrypts the encoded code s using the symmetric password-based encryption PBE.enc, the posterior probability of a vault v is obtained by aggregating probabilities of all cases, i.e.,

$$\frac{\Pr_{\mathrm{V}}(v)}{\Pr_{\mathrm{C}}(c)} \sum_{s\in\mathcal{S}} \frac{1}{|\mathcal{S}|\cdot \Pr_{\mathrm{D}}(v)} \sum_{mp\in\mathcal{MP}} \Pr_{\mathrm{MP}}(mp)\Pr(\mathsf{enc}(mp,s)=c). \quad (4)$$

We also adopt the following two realistic assumptions in [25] to simplify the computation of Bayesian posterior probability:

- For any vault v and its ciphertext c, the event that only one master password mp and one code $s \in \mathcal{S}$ can make $\mathsf{enc}(mp,s)=c$ hold with an overwhelming probability, i.e., $\sum_{s\in\mathcal{S}} \sum_{mp\in\mathcal{MP}} \Pr_{\mathrm{MP}}(mp)\cdot\Pr(\mathsf{enc}(mp,s)=c) \approx \Pr_{\mathrm{MP}}(mp)$.
- Passwords in the vault and the master password are chosen by the same user under the same password creation behaviors. Thus, the strength of the master password is consistent with the user's real vault, i.e., $\Pr_{\mathrm{MP}}(mp) \approx \Pr_{\mathrm{V}}(v)$ for normalized $\Pr_{\mathrm{MP}}(mp)$ and $\Pr_{\mathrm{V}}(v)$ (e.g., n=1,000 vaults).

Under the above premises, we can give the score function of the theoretically-grounded distinguishing attack [25] as

$$\mathrm{s}_{\mathrm{TG}}(v) \coloneqq \frac{\Pr_{\mathrm{V}}(v)}{\Pr_{\mathrm{D}}(v)} \cdot \Pr_{\mathrm{MP}}(mp) = \frac{\Pr_{\mathrm{V}}(v)^2}{\Pr_{\mathrm{D}}(v)}. \quad (5)$$

For theoretical clarity, in this part, we suppose both the NLE and the attacker $\mathcal{A}$ can ideally know the user's real password vault distribution. To begin with, once the n vaults have been given, there is only a constant coefficient difference (i.e., $\frac{\sum_{i=1}^{n} \Pr_{\mathrm{D}}(v_i)}{\sum_{i=1}^{n} \Pr_{\mathrm{V}}(v_i)^2}$) between $\mathrm{s}_{\mathrm{TG}}(\cdot)$ obtained from normalized and unnormalized probabilities. Thus, when calculating score functions, we don't discriminate if $\Pr_{\mathrm{V}}(v)$ and $\Pr_{\mathrm{D}}(v)$ are normalized. Other distinguishing attacks in Appendix A can be analyzed in similar approaches. Based on these premises, we present Theorem 1 as the first theoretical contribution.

Theorem 1. *For any user that has one real and arbitrary $n-1$ decoy vaults (with $n<|\mathcal{V}|$, where $\mathcal{V}$ is the vault space), when faced with Duan et al.'s theoretically-grounded distinguishing attack [25], the NLE can reduce this attack to uniform random guesses, i.e., making $\mathrm{s}_{\mathrm{TG}}(v)=\frac{\Pr_{\mathrm{V}}(v_i)^2}{\Pr_{\mathrm{D}}(v_i)} \equiv C$ constant for all $v_i \in \{v_1 \ldots, v_n\}$ if and only if the normalized $\Pr_{\mathrm{D}}(v_i)=\Pr_{\mathrm{V}}(v_i)=\frac{1}{n}$ holds. Otherwise, the best an NLE can do is to force $\mathcal{A}$ to guess in the descending order of $\Pr_{\mathrm{V}}(v)$.*

The detailed proof is shown in Appendix B.1. In short, the optimality of $\mathcal{A}$'s and the NLE's strategies are guaranteed by Bayesian estimation and Stackelberg equilibrium. This theorem reveals that unless the real and decoy vaults are all uniformly distributed, e.g., passwords in the real vault are randomly generated and decoys are generated via Chatterjee et al.'s UNIF [17], the most desirable case that no vault is more likely to be real is unachievable. *Due to Zipf's nature of human-generated passwords [64], the NLE cannot reduce the theoretically-grounded attack [25] to uniform guesses.* In this case, the best an NLE can do is to set $\Pr_{\mathrm{D}}(v)=\Pr_{\mathrm{V}}(v)$, i.e., the PPM adopted by the NLE should be as close to the real vault distribution as possible. We call NLEs that satisfy this condition resisting the theoretically-grounded attack [25], and give the formal definition as follows. Such NLEs provide $\mathcal{A}$ with no additional information, and force her to guess based on the known/approximated $\Pr_{\mathrm{V}}(v)$.

Definition 2. *An NLE is called to resist Duan et al.'s theoretically-grounded distinguishing attack [25] if it can force $\mathcal{A}$ to guess in the descending order of $\Pr_{\mathrm{V}}(v)$. An NLE is called to resist other distinguishing attacks if it can force $\mathcal{A}$ to guess uniformly at random, i.e., the score functions $\mathrm{s}(v)$ are about the same for any v.*

We also compare ours with Duan et al.'s two score functions of the theoretically-grounded attack $\mathrm{s}_1(v)=\prod_{pw\in v} \frac{\Pr_{\mathrm{V}}(pw)^2}{\Pr_{\mathrm{D}}(pw)}$ and $\mathrm{s}_2(v)=\frac{\Pr_{\mathrm{V}}(v)^{\frac{n+1}{n}}}{\Pr_{\mathrm{D}}(v)}$ [25]. First, $\mathrm{s}_1(\cdot)$ is a special case of Eq. 5, i.e., $\mathrm{s}_{\mathrm{TG}}(v)=\frac{\Pr_{\mathrm{V}}(v)^2}{\Pr_{\mathrm{D}}(v)}$, since they are identical if all passwords in a vault are independently generated. Second, the following Corollary 1 (see the proof in Appendix B.1) reveals that $\mathrm{s}_2(\cdot)$ is equivalent to Eq. 5: An NLE that can resist the theoretically-grounded attack [25] instantiated under $\mathrm{s}_{\mathrm{TG}}(v)$ can also resist that instantiated under $\mathrm{s}_2(v)$, and vice versa, which strengthens using $\mathrm{s}_{\mathrm{TG}}(v)$.

Corollary 1. *An NLE can resist the theoretically-grounded attack instantiated under $\mathrm{s}_{\mathrm{TG}}(v)=\frac{\Pr_{\mathrm{V}}(v)^2}{\Pr_{\mathrm{D}}(v)}$ (resp. $\mathrm{s}_2(v)=\frac{\Pr_{\mathrm{V}}(v)^{\frac{n+1}{n}}}{\Pr_{\mathrm{D}}(v)}$) can also resist that instantiated under $\mathrm{s}_2(v)$ (resp. $\mathrm{s}_{\mathrm{TG}}(v)$) by forcing $\mathcal{A}$ to guess in the same order: For any two vaults $v_i, v_j \in \{v_1, v_2, \ldots, v_n\}$, $\mathrm{s}_2(v_i) \geq \mathrm{s}_2(v_j)$ if and only if $\mathrm{s}_{\mathrm{TG}}(v_i) \geq \mathrm{s}_{\mathrm{TG}}(v_j)$.*

Table 1: Overview of existing vulnerabilities and honey vault design criteria.[†]

Type of vulnerability	Corresponding attack	Honey vault design criteria	Our VaultGuard solution
Difference between real and decoy vaults	KL divergence attack [29], single password attack [19], password similarity attack [19], theoretically-grounded attack [25]	**Criterion #1**: PPMs should accurately model users' real-world password distributions for both unique passwords and reused passwords in the vault.	Employing accurate sequence -based UPPMs resisting the theoretically-grounded attack [25] and making RPPMs ambiguity-free (see Sec. 4.1)
Ambiguous password generation rules	Weak encoding attack[20], strong encoding attacks [20]	**Criterion #2**: Segment-based UPPMs should not be used in building NLEs; RPPMs should be sufficiently concise with no fixed standard for selecting rules.	
Publicity of adaptive PPMs employed by users	Adaptive extra attack [19], adaptive hybrid attack [19]	**Criterion #3**: Users' adaptive PPMs should be inaccessible to attackers trying incorrect master passwords.	Employing our HE-Adaptive mechanism (see Sec. 4.2)

[†] PPM denotes a password probability model, including the unique model (i.e., UPPM) for unique passwords, and the reuse model (i.e., RPPM) for reused ones. Ambiguous password generation rules denote different rules parsing the same password. For instance, in PCFG-based [72] NoCrack-NLE [17], "`password`" can be seen as either a single word segment "`password`" or a combination of two different word segments "`pass`" and "`word`".

Reduction of distribution-aware attacks. For an NLE resisting the theoretically-grounded attack [25], we formalize the condition $\Pr_D(v)\approx\Pr_V(v)$ as $|\frac{\Pr_V(v)-\Pr_D(v)}{\Pr_D(v)}|<\epsilon$ with a sufficiently small ϵ (e.g., $\epsilon\approx 0$). Namely, for a password $pw\in v$, there is $\left|\frac{\Pr_{PW}(pw)-\Pr_D(pw)}{\Pr_D(pw)}\right|<\epsilon$, where $\Pr_{PW}(\cdot)$ is the real-world password probability. Under these premises, we have Theorem 2 as follows for Criterion #1.

Theorem 2. *If an NLE can resist Duan et al.'s theoretically-grounded distinguishing attack [25] whose score function is Eq. 5, it can also resist Golla et al.'s KL divergence attack [29], and Cheng et al.'s single password attack and password similarity attack [19].*

Proof. The score function of the KL divergence attack is $s_{KL}(v)=\sum_{pw\in v}\Pr_f(pw)\cdot\log\left(\frac{\Pr_f(pw)}{\Pr_D(pw)}\right)$ [29], where $\Pr_f(pw)$ is the relative frequency of $pw\in v$ and $\Pr_D(pw)$ is the probability of pw in decoy vaults sampled by the NLE (see Appendix A). For $\mathcal{A}$ and the NLE who know $\Pr_V(v)$, there is $\Pr_f(pw)\approx\Pr_{PW}(pw)$. Thus, we have

$$\begin{aligned} s_{KL}(v) &= \sum_{pw\in v}\Pr_f(pw)\log\left(\frac{\Pr_f(pw)}{\Pr_D(pw)}\right)\leq\sum_{pw\in v}\left(\frac{\Pr_f(pw)}{\Pr_D(pw)}-1\right)^2 \\ &\approx \sum_{pw\in v}\left(\frac{\Pr_{PW}(pw)}{\Pr_D(pw)}-1\right)^2<|v|\epsilon^2, \end{aligned} \tag{6}$$

where the first inequality comes from the χ^2 divergence inequality [22] (see details in Lemma 1 of Appendix B.1). *Hence, for all vaults, their scores are uniformly lower than $|v|\epsilon^2$ (close to 0), so the KL divergence attack [29] is degraded to guessing uniformly at random.*

For Cheng et al.'s single password attack with the score function $s_{SP}(v)=\frac{\Pr_V(v)}{\Pr_D(v)}=\prod_{pw\in v}\frac{\Pr_{PW}(pw)}{\Pr_D(pw)}$ [19] (see Appendix A), we have

$$s_{SP}(v)=\prod_{pw\in v}\frac{\Pr_{PW}(pw)}{\Pr_D(pw)}<(1+\epsilon)^{|v|}\approx 1+|v|\cdot\epsilon, \tag{7}$$

so scores given by Cheng et al.'s single password attack [19] are uniformly no more than $1+|v|\cdot\epsilon$ (close to 1). Thus, the single password attack [19] is also degraded to uniform guesses. For password similarity attack [19], an ideal NLE can generate (almost) all similarity features of real vaults, so there is $|\frac{\Pr_V(F=F(v))-\Pr_D(F=F(v))}{\Pr_D(F=F(v))}|<\epsilon$ for features $F\in\mathcal{F}$, and we can have the result accordingly. □

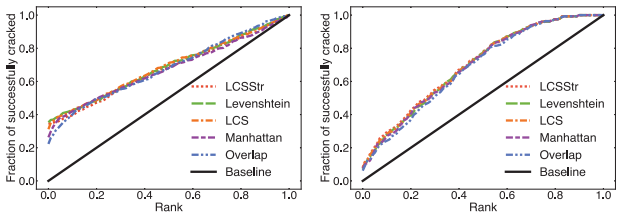
Remark 1. We have proved that an NLE resisting the theoretically-grounded attack [25] can also resist other distribution-aware distinguishing attacks, *but the opposite is not true*. We provide a toy example to show this point. Suppose the probabilities of the four unique passwords in the vault v are $(\Pr_{PW}(pw_1),\ldots,\Pr_{PW}(pw_4))=(0.0001, 0.3333, 0.3333, 0.3333)$, but those of the NLE are $(\Pr_D(pw_1),\ldots,\Pr_D(pw_4))=(0.000001,\ 0.333333,\ 0.333333,\ 0.333333)$. In this case, we have $s_{KL}(v)=0.000362\approx 0$, while $\frac{\Pr_V(v)}{\Pr_D(v)}=\prod_{i=1}^{4}\frac{\Pr_{PW}(pw_i)}{\Pr_D(pw_i)}=99.97$, i.e., $\Pr_V(v)\gg\Pr_D(v)$. Thus, the vault is secure against the KL divergence [29], but NOT the theoretically-grounded attack [25].

Remark 2. We show the real-world implications of our theories. In practice, neither $\mathcal{A}$ nor the NLE explicitly knows the real vault or password distributions. Instead, they employ various state-of-the-art PPMs, including unique and reused password probability models to approximate/instantiate real ones. Based on Theorem 1, the NLE should make the decoy vault distribution $\Pr_D(v)$ as close to $\mathcal{A}$'s instantiated $\Pr_V(v)$ as possible, so it can resist distinguishing attacks that exploit the differences between real and decoy distributions. This reveals the need to investigate state-of-the-art PPMs (e.g., [40, 41, 47, 70]), and we present our experimental results in Sec. 4.1.

Security analysis of reuse models. Existing honey vault studies (see [17, 19, 25, 29]) mainly focus on how unique password models such as PCFG [72], Markov [40] impact distribution-aware distinguishing attacks, while largely leaving users' reuse behaviors untouched. To the best of our knowledge, current security analysis of reuse behaviors mainly focuses on: (1) encoding attacks that exploit ambiguities in RPPMs, which have little relevance with vault distributions; (2) simply enumerating similarity features in real and decoy vaults without considering real and decoy vault distributions. In the following part, we show how such a minor difference can cause catastrophic consequences in evaluating the NLE security.

To show our point, we first present our key observations of differences between real-world reuse behaviors and existing RPPMs. First, Golla et al.'s RPPM [29] only considered reusing *one* password, so it cannot generate vaults of the form $v=(pw_A, pw_{A'},\ldots, pw_B, pw_{B'},\ldots)$, where $pw_{A'}$ and $pw_{B'}$ are modified from pw_A and pw_B respectively. Second, though Cheng et al.'s RPPM [19] considered the possibility of reusing multiple passwords, it assumed passwords in the tail part of a vault are more likely to be reuses of the prior ones [19]. Thus, vaults of the form $(pw_A, pw_{A'},\ldots, pw_{A''}, pw_B,\ldots)$ are unlikely to be generated by Cheng-NLE [19]. However, in practice, both reusing multiple passwords and creating unique passwords for later registered important accounts (e.g., financial) are common. Besides, the observed site sequences in a real vault (e.g., `Pastebin`) are not necessarily equal to the registration order.

Based on these key observations, we now provide a simple yet effective attack against existing RPPMs used in Golla-NLE [29] and Cheng-NLE [19]. First, we traverse the vault to find all unique passwords and group reused passwords modified from the same unique password into one class. We use the five password similarity features in [19] (see details in Table 4 in Appendix C) to determine if two passwords are reused. In each class, $\Pr_V(v)$ of reused passwords are instantiated by TarGuess-II [67] as in [25], while unique ones are



(a) Reuse attack against Golla-NLE [29] (b) Reuse attack against Cheng-NLE [19]

Figure 2: Experimental results of our attack against RPPMs of Golla-NLE [29] and Cheng-NLE [19] under varied similarity features in [19]. The black baseline curve represents guessing vaults uniformly at random. The results reveal that existing RPPMs are insecure.

treated as uniformly distributed. The decoy distributions $\mathrm{Pr}_{\mathrm{D}}(v)$ are instantiated as Golla-NLE [29] and Cheng-NLE [19], and the score function is $\mathrm{s}_{\mathrm{RP}}(v)=\frac{\mathrm{Pr}_{\mathrm{V}}(v)^2}{\mathrm{Pr}_{\mathrm{D}}(v)}$ as Eq. 5. Fig. 2 shows that the CDGs, and the average rank $\bar{r}$ against Golla-NLE [29] and Cheng-NLE [19] are 31.21%∼32.73% and 29.75%∼31.73% (with the ideal $\bar{r}$=50%), respectively. Compared with Cheng et al.'s password similarity attack [19], our attack requires 33% fewer online verifications.

3.3 Analysis on Criterion #2

Criterion #2 aims to resist weak and strong encoding attacks [20] arising from ambiguous password generation rules. Generically, NLEs based on segment-based UPPMs (e.g., PCFG-based NoCrack-NLE [17]) fix a standard (i.e., the highest probability) for selecting from multiple rule vectors that generate the same segment. In this way, a vault v comprising an ambiguous password segment in pw^* is parsed into a rule vector $\mathbf{rv}_v$ comprising the specific $\mathbf{rv}_{pw^*}$, i.e., $\mathbf{rv}_v=(\mathbf{rv}_{pw_1},\ldots,\mathbf{rv}_{pw^*},\ldots,\mathbf{rv}_{pw_{|v|}})$. Thus, *any decoded vault comprising pw^* but not rule vectors comprising $\mathbf{rv}_{pw^*}$ is a decoy.* Cheng et al.'s exploited this feature, and proposed weak and strong encoding attacks [20]: For the weak encoding attack, $\mathrm{s}_{\mathrm{WK}}(v)$=1 if the encoded rule vector is equal to the decoded one, and 0 otherwise; For the strong encoding attack, $\mathrm{s}_{\mathrm{SG}}(v)=\frac{\mathrm{s}_{\mathrm{WK}}(v)}{\mathrm{Pr}_{\mathrm{D}}(v)}$.

We reveal that: (1) *NLEs based on segment-based UPPMs are inherently vulnerable to Cheng et al.'s encoding attacks [20]; (2) Cheng et al.'s solution [20] is vulnerable to distribution-aware attacks; (3) Cheng et al.'s solution [20] suffers from exponential time complexity regardless of instantiations.* For the first point, we replace PCFG [72] with PwdSegment [76] for NoCrack-NLE [17], and conduct weak and strong encoding attacks. Fig. 3 shows high attack efficiencies with CDGs far above the baseline, and the average ranks of weak and strong attacks are 20.4% and 3.9%, respectively. This substantiates that NLEs built on segment-based UPPMs are inherently vulnerable to encoding attacks.

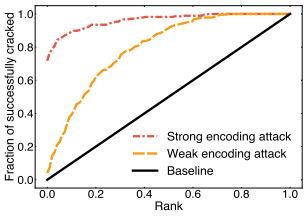


Figure 3: Experimental results of NoCrack-NLE [17] instantiated on PwdSegment [76] against the weak and strong encoding attacks [20]. This reveals the effectiveness of generic encoding attacks.

For the second point, we show how Cheng et al.'s IS-PMTE [20] interacts with distribution-aware attacks. First, in encoding, IS-PMTE [20] enumerates all rule vectors $\mathbf{rv}_v$ when parsing the user's real vault v (the set of all such rule vectors is denoted as $\mathbf{RV}(v)$), and randomly selects one based on its probability/weight in $\mathbf{RV}(v)$. Thus, the security of an IS-PMTE [20] against distribution-aware attacks is upper bounded by the corresponding NLE that chooses rule vectors with the highest probabilities in parsing ambiguous password segments. We explain this point as follows. When the rule vector of an ambiguous segment is sampled based on its probability rather than deterministically chosen as the most probable one, in decoding, the decoy probability $\mathrm{Pr}_{\mathrm{D}}(v)$ of the real vault will decrease. However, $\mathrm{Pr}_{\mathrm{D}}(v)$ for the n−1 decoy vaults will remain the same, since they are decoded as rule vectors from codes before parsing. Since $\mathrm{Pr}_{\mathrm{V}}(v)$ is instantiated by $\mathcal{A}$, it remains the same. In this way, the real vault will be ranked lower and guessed first.

The above reasoning enables us to apply the results of distribution-aware attacks against original NLEs to those of IS-PMTEs. We do experiments on NoCrack-NLEs and IS-PMTE under PCFG [72] and PwdSegment [76] grammars. For PCFG-instantiated ones, the average ranks against distribution-aware attacks (where the theoretically-grounded attack is instantiated with the List model and TarGuess-II as in [25]) range from 12.98%∼48.87%, while those of IS-PMTE are 12.31%∼32.60%. The small difference is because the highest probability rule vector often takes an overwhelming weight. For instance, in the Rockyou trained PCFG model, $\mathrm{Pr}_{\mathrm{PW}}(\mathtt{password})$=0.005, while $\mathrm{Pr}_{\mathrm{PW}}(\mathtt{pass})\cdot\mathrm{Pr}_{\mathrm{PW}}(\mathtt{word})\approx 10^{-8}$. PwdSegment-based NoCrack-NLE and IS-PMTE share similar results, and their average ranks range from 21.16%∼32.06%. Hence, IS-PMTE is inherently insecure.

To show the third point, we provide an improved Cheng et al.'s IS-PMTE [20], and prove that its complexity is still exponential. More concretely, our improved IS-PMTE works as follows.

- **Step #1**. The generic IS-PMTE parses passwords in the vault into disjoint types of segments. Then, it finds rule vectors of all subsegments that can be directly parsed by one UPPM grammar (i.e., single rules) of each type, and their probability $\mathrm{Pr}_{\mathrm{D}}(\cdot)$ values (see Definition 3 and Alg. 1 in Appendix B.2).
- **Step #2**. For each segment in the vault, the generic IS-PMTE combines all its single rules into rule vectors and computes their probabilities (see Alg. 2 in Appendix B.2).
- **Step #3**. The generic IS-PMTE combines rule vectors of segments to obtain rule vectors of passwords and the vault. Each rule vector of the vault is selected based on its probabilities in the rule vector set $\mathbf{RV}(v)$ (see Alg. 3 in Appendix B.2).

Once the rule vector $\mathbf{rv}_v$ has been chosen, the vault v can be readily encoded/decoded into code/vaults through IS-DTE [36]. Built on these concrete steps, we derive Theorem 3 that provides a quantitative time complexity for IS-PMTE-style solutions.

Theorem 3. *Under the premises of segment-based UPPMs in Sec. 2.2, the time complexity of finding all rule vectors parsing the vault via Steps #1∼#3 is $O\Big(\sum_{m,i,j}(\sum_{l=1}^{l_{m,i,j}} n_{m,l}+2^{l_{m,i,j}})\Big)$, where $n_{m,l}$ is the number of the length l and type-m segments in the UPPM grammars.*

The proofs are shown in Appendix B.2. Our scheme improves the IS-PMTE time complexity from $O(\prod_{m,i,j}2^{l_{m,i,j}})$ in [20] to $O(\sum_{m,i,j}2^{l_{m,i,j}})$ in this work. No ambiguity exists in sequence-based UPPMs as passwords are sequentially generated in a character-wise manner.

Remark 3. The above methodology can also be applied to RPPMs, i.e., no standard should be fixed for modifying reused passwords.

Additionally, the number and types of modifications should also be limited. In Appendix C, we explain why Cheng et al.'s RPPM [19] is vulnerable to encoding attacks [20] that it aims to resist.

3.4 Analysis on Criterion #3

Criterion #3 aims to address the inherent conflict in simultaneously resisting distribution-aware (see [19, 25, 29]) and adaptive extra [19] attacks, and this task is inherently difficult. On the one hand, if the user's PPM is uncorrelated with the real vault, decoy vaults are dissimilar to the real vault, and thus vulnerable to distribution-aware attacks [29]. On the other hand, an adaptive PPM correlated with the real vault facilitates adaptive extra and hybrid attacks [19]. *To our knowledge, there is no solution to this long-standing problem.*

To more explicitly show the conflicts, we first formalize the adaptive mechanism as the tuple $\beta=(\theta, \Theta, p_\theta, k_C)$, where θ is the set of all selected d-grams sampled from Θ according to the distribution p_θ, k_C instructs how these d-grams are boosted. For example, in Golla et al.'s mechanism [29] (see Sec. 2.2), θ denotes the set of all selected d-grams (e.g., d=4), Θ is its probability space given by a Markov UPPM [40], and p_θ is the combination of: (1) uniformly selecting one d-gram from each password in the real vault (the number is $|v|$); (2) selecting from all d-grams except the $|v|$ ones in the first step based on the binomial distribution $B(|\Theta|-|v|, p)$, where p=20% [29]. The boost parameter k_C=5, i.e., the frequencies of selected 4-grams are multiplied by 5. Thus, in Golla et al.'s mechanism [29], θ and p_θ can be seen as random functions of the user's real vault v. We mainly focus on adaptive UPPMs, and the case of RPPM is similar.

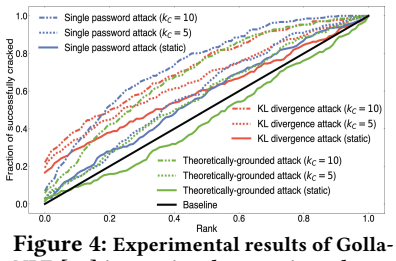


Figure 4: Experimental results of Golla-NLE [17] instantiated on static and randomly adaptive PPMs uncorrelated with the user's real vault. We find that against our instantiated theoretically-grounded [25], KL divergence [29] and single password [19] attacks. The larger the k_C value, the smaller the average ranks.

To show that θ and p_θ need to rely on the real vault to resist distribution-aware attacks, we use the idea of proof by contradiction. Particularly, for Golla-NLE [29], we set all boosted 4-grams randomly selected based on the binomial distribution p_θ=$B(|\Theta|, p)$ regardless of whether they are in the real vault. We then generate 1,000 such PPMs, and evaluate the security under our instantiated theoretically-grounded attack (i.e., Eq. 5), KL divergence [29] and single password [19] attacks, and compare the results with those of the static PPM. *The CDGs in Fig. 4 show that $\mathcal{A}$'s attack efficiencies have been significantly enhanced for such adaptive PPMs.* Besides, the average ranks become 6.84%∼42.43% smaller, and the larger the k_C value, the smaller the average ranks. Similar results also happen for other p and d-gram settings. This is expected, as such adaptive PPMs uncorrelated with the real vault can make unpopular 4-grams undesirably more popular, and make PPM less accurate.

The above results reveal the necessity to make the user's adaptive PPM θ correlated with the real vault v to generate more similar decoy vaults. However, following the Kerckhoffs's principle, once a honey vault is leaked, the attacker $\mathcal{A}$ can know θ, and compare the static and adaptive PPMs to identify boosted d-grams, which enables $\mathcal{A}$ to extract information related to the real vault from these d-grams (i.e., adaptive extra attack [19], see Appendix A). Since the adaptive PPM is correlated with the real, not any decoy vault, extraction attacks are effective. Hence, a secure honey vault scheme needs to make a user's adaptive PPM (correlated with the real vault) inaccessible to attackers trying incorrect master passwords.

4 Our VaultGuard Scheme

Our proposed new honey vault scheme is comprised of (1) VaultGuard-NLE to meet Criteria #1&#2, and (2) the HE-Adaptive to meet Criterion #3 (see Table 1). First, VaultGuard-NLE employs the accurate *sequence-based* 5-gram Markov UPPM [40] , and an enhanced RPPM. By our designs, VaultGuard-NLE is resistant to distribution-aware and weak and strong encoding attacks. Second, we treat boosted d-grams of the user's adaptive PPM as the secret, and propose a systematic solution named HE-Adaptive to honey-encrypt a user's adaptive PPM. Through our HE-Adaptive, each pair of a decoy vault and its adaptive PPM used for encoding/decoding vaults maintains the same relation as the real ones to resist various extraction (e.g., adaptive extra&hybrid [19]) attacks.

4.1 Our VaultGuard-NLE design

Following Criteria #1&#2, we investigate state-of-the-art sequence-based UPPMs and RPPMs to instantiate $\Pr_D(v)$. For UPPMs, we consider statistics-based Markov and Backoff [40], machine-learning-based RFGuess [70], and deep-learning-based FLA [41] and dynamic password guessing (DPG) [47] models. Further, to balance security and usability, we adopt 5-gram UPPMs (i.e., d=5). This is because a PPM needs to be stored on the client side (e.g., a laptop) to make an NLE work, and large d values take undesirably large storage spaces. Our preliminary experiments show that a 5-gram UPPM takes about 15.7MB, but a 6-gram UPPM requires over 200MB, far exceeding off-the-shelf password manager extensions that take 136KB∼86MB (see Table 5 in Appendix D). Hence, Cheng et al.'s 6-gram Markov UPPM [19] is impractical for deployment. For training, as stated in Sec. 2.4, we train UPPMs with default settings. Particularly, for DPG [47] that does not generate password probabilities, we run the trained DPG model for 10^9 times and sample password frequencies to estimate their probabilities by the law of large numbers.

For RPPM, we exclude complicated models like TarGuess-II [67], Pass2Path [46], Pass2Edit [69] and PassBert [77] that bring extensive ambiguities [19]. In particular, we use Pass2Edit [69] that has not been discussed in [19, 20, 25, 29] as an example. Pass2Edit [69] only allows deletions and insertions, and adopts the minimum edit distance modifications when multiple edit approaches exist. This strategy is similar to the PCFG-based NoCrack-NLE [17] that adopts the rule vector with the highest probability, so ambiguities exist. Besides, even if the edit distance is at most 5 and the password length is ≤20, the space size of all modifications can be as large as $(20 \times 94 + 20)^5 \approx 10^{16}$. Here, 94 is the number of all printable ASCII characters, 20×94 is the number of possible insertions, and 20 is the number of deletions. Thus, the IS-PMTE-style solution in Sec. 3.3 is impractical. Similar problems also exist in PassBert [77].

We design our RPPM to address this issue (see details in Appendix C). Following [20, 24, 63], we mainly focus on encoding and decoding character modifications in the head and tail parts, and

use the LCSStr feature as [20] (i.e., the longest common string is over half the maximum length) to determine if two passwords are reused. Our RPPM adopts delete, insert, and delete-then-insert modifications in [20], *without any of their combinations. Unlike Cheng et al.'s RPPM [20], ours does not specify which modifications are used in advance, but only the number of each type of atomic modification (i.e., deleting/inserting one character), to resist the encoding attacks.* Our RPPM also groups reused passwords of the same unique one into one class, and different classes are disjoint. We take the conservative assumption that attackers know the indices of each unique password and all passwords reused from it, and only do not know how passwords are modified. For more relaxed cases, these indices can be treated as secrets and protected by honey encryption, which can be designed and implemented accordingly. This overcomes the drawback entailed by Cheng et al.'s assumption that passwords in the tail of a vault are more likely to be reused [20].

As shown in Sec. 3.2, we use the theoretically-grounded attack [25] in the form of $\frac{\Pr_V(v)^2}{\Pr_D(v)}$ (i.e., Eq.5) to evaluate the security of VaultGuard-NLE. As in [25], the UPPM in $\Pr_V(v)$ for unique passwords is instantiated under the List model [68] that enumerates password probabilities in a dataset; For RPPM, we assume that reused passwords are modified independently, and instantiate each of them in $\Pr_V(v)$ as TarGuess-II [67]. As shown in Fig. 5, the CDGs of Markov [40] and FLA [41] are the closest to the baseline. The average rank also shows this result, with the values under Markov [40], Backoff [40], RFGuess [70], FLA [41] and DPG [47] being 35.24%, 27.69%, 32.46%, 35.00% and 29.65%, respectively. Though FLA is also close, it is essentially the 10-gram Markov [41], so it is computationally impossible to make it adaptive. Hence, we mainly use the 5-gram Markov UPPM [40] for our VaultGuard-NLE.

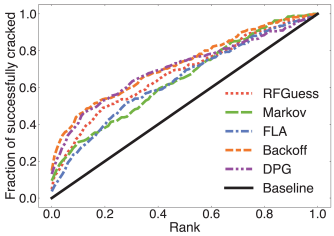


Figure 5: Experimental results of our VaultGuard-NLE instantiated on Markov [40], Backoff [40], RFGuess [70], FLA [41] and DPG [47] sequence-based UPPMs along with our RPPM (see Appendix C) against the theoretically-grounded attack [25]. The green curve represented by the Markov UPPM [40] is the closest to the baseline.

4.2 Our HE-Adaptive mechanism

Although honey-encrypting the user's adaptive PPM seems promising to achieve Criterion #3, this task is challenging for two reasons. First, inherent conflicts exist. On the one hand, decoding an adaptive PPM θ requires the explicit adaptive PPM distribution p_θ, which requires the vault to be decoded first (see Sec. 3.4); On the other hand, decoding vaults requires θ to be decoded first under the standard honey encryption workflow (see Sec. 2.1). *This puts honey-encrypting adaptive PPMs into a chicken-and-egg situation.* Second, a sequence-based UPPM is created from the leftmost to the rightmost (see Sec. 2.2). Hence, for each selected d-gram in a decoy vault, the adaptive PPM can only calculate probabilities of the d-grams after (i.e., forward) it, but not the ones before (i.e., backward). To generate indistinguishable decoy vaults, it is necessary to derive backward d-gram probabilities from the forward ones.

Our HE-Adaptive addresses these challenges in a principled approach. At a high level, for the first challenge, our HE-Adaptive treats (1) the position of each selected d-gram of each password and (2) the d-gram content as secrets, and honey-encrypts them. Positions are encoded under a uniform distribution related to the maximum password length. For d-grams, the selected ones are encoded under the *static PPM*, while others in the real vault are encoded under the *adaptive PPM* boosted (e.g., multiplying current probabilities/frequencies by k_C) by the selected d-grams. For the second challenge, HE-Adaptive exploits the internal dependence between forward and backward d-grams, and reconstructs backward probabilities from the forward ones in an efficient approach.

HE-Adaptive encoding process comprises three major steps. For simplicity, we take d=5 as in Sec. 4.1, and other cases are similar. We also denote 5-grams of the form (abcd,x) in [29] as the forward 5-gram, since an undetermined x (ranging from all 94 printable ASCII characters) is positioned after the known string abc. Similarly, 5-grams of the form (y,abc) are denoted as backward 5-grams.

With the above notations, in Step #1, HE-Adaptive encodes the positions of selected 5-grams. To do this, we treat the position of the first character in a 5-gram as its position in a password. For example, the position of (pple,1) in "apple123" has pos=2. In this manner, the number of pos values in a password equals that of 5-grams, i.e., at most l_M-3 in our secure incremental update (IU) mechanism (where l_M is the maximum length of a password in the vault, see details in Appendix C), and a pos value is uniformly selected. Thus, HE-Adaptive can treat pos as the message and $U[1, l_M-3]$ is its distribution, and follow the standard honey encryption workflow in Sec. 2.1 to encode messages through IS-DTE [36].

In Step #2, each selected 5-gram is encoded under the static PPM like *static* Golla-NLE [29]. Thus, we mainly focus on encoding unselected 5-grams and positions of selected 5-grams. For computation simplicity, unlike Golla et al.'s adaptive mechanism [29], HE-Adaptive boosts the revised 5-grams (denoted as 5-rvgrams) of the selected ones with the form (abcd$x$), rather than (abcd, x)=x|abcd in [29], where $\Pr_{PW}(\texttt{abcd}, x)=\frac{\texttt{Count(abcd}x\texttt{)}}{\texttt{Count(abcd)}}$ and $\texttt{Count}(\cdot)$ counts frequencies of 5-rvgrams in the static PPM. In this way, forward 5-gram (abcd,x) probabilities in the adaptive PPM are:

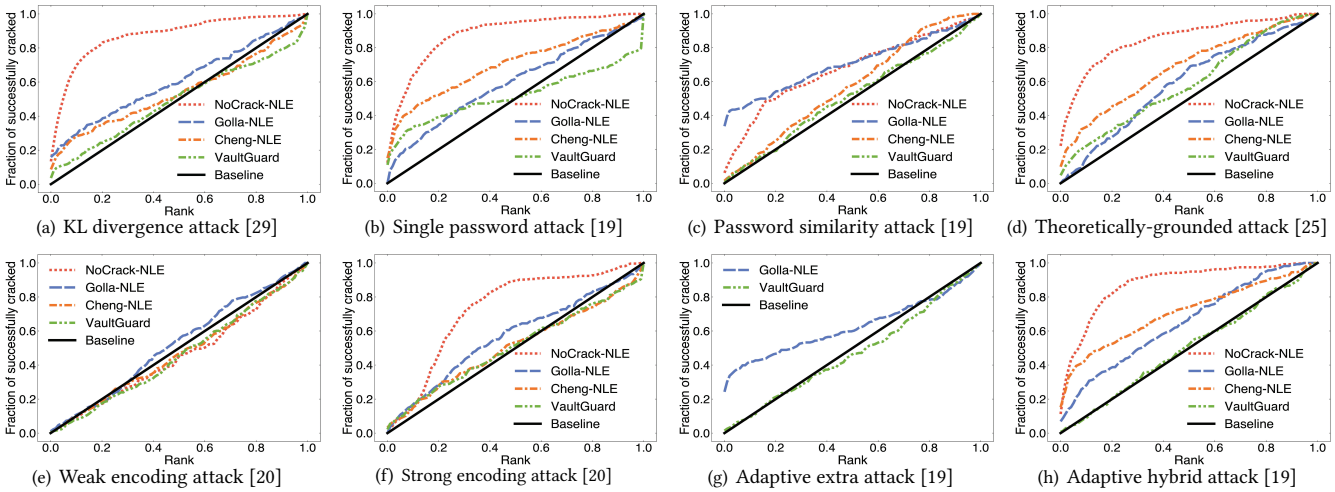
$$\Pr_{PW}(\texttt{abcd}, x) \coloneqq \frac{k'_C \cdot \texttt{Count(abcd}x\texttt{)}}{\sum_{x'} k'_C \cdot \texttt{Count(abcd}x'\texttt{)}}, \tag{8}$$

where k'_C=k_C if the 5-rvgram (abcd$x$) is boosted, and k'_C=1 otherwise. For each password, unselected 5-grams positioned after the selected one are encoded under the adaptive PPM given by Eq. 8.

As the positions and contents of selected 5-grams are encoded, in Step #3, HE-Adaptive encodes backward 5-grams before the selected one in a password to finish encoding the vault. To do this, we exploit the relation that y occurring before abcd is equivalent to that d occurring after $y$abc, i.e., (y, abcd)=($y$abc, d), and there is

$$\Pr_{PW}(y, \texttt{abcd}) \coloneqq \frac{\Pr_{PW}(y\texttt{abcd})}{\Pr_{PW}(y\texttt{abc})} \Big/ \sum_{y'} \frac{\Pr_{PW}(y'\texttt{abcd})}{\Pr_{PW}(y'\texttt{abc})}, \tag{9}$$

where $\Pr_{PW}(\cdot)$ is the probability in the adaptive PPM, and $\sum_{y'} \frac{\Pr_{PW}(y'\texttt{abcd})}{\Pr_{PW}(y'\texttt{abc})}$ is used for normalization. Thus, all backward 5-grams before the selected ones are encoded under the distribution given by Eq. 9. While the code in Step #1 is stored separately, the code of the real vault is obtained by concatenating codes in Steps #2&#3.



(a) KL divergence attack [29] (b) Single password attack [19] (c) Password similarity attack [19] (d) Theoretically-grounded attack [25]

(e) Weak encoding attack [20] (f) Strong encoding attack [20] (g) Adaptive extra attack [19] (h) Adaptive hybrid attack [19]

Figure 6: Security evaluation results (in the CDG form) of honey vault (HV) schemes (i.e., NLEs) against all existing HV distinguishing attacks. VaultGuard in this figure means our VaultGuard-NLE with HE-Adaptive. Golla-NLE means the adaptive 5-gram Golla-NLE with the parameters k_C=5 and p=20% as in [29]. Since adaptive extra and hybrid attacks target adaptive sequence-based PPMs, we do not use them to attack VaultGuard-NLE, NoCrack-NLE [17] and Cheng-NLE [19]. Evaluation results reveal that in general, our VaultGuard outperforms existing HV schemes.

HE-Adaptive decoding process. When an incorrect master password is tried, the position *pos* will be first decoded into another integer following $U[1, l_M-3]$. Then, each selected forward 5-gram, whose position is obtained by the decoded integer, will be decoded under the static PPM. The rest 5-grams are decoded under the adaptive PPM boosted by the corresponding 5-rvgrams. Finally, backward 5-grams are decoded under the adaptive PPM, and a decoy vault is generated. In this way, an adaptive PPM has one 5-gram of each password in a decoy vault, so the relation between the real vault and the adaptive PPM can be kept. This provides $\mathcal{A}$ with no extra advantages in distinguishing between real and decoy vaults.

Remark 4. Unlike Golla et al.'s adaptive mechanism [29], our HE-Adaptive does not consider d-grams that are boosted based on the binomial distribution $B(|\Theta|-|v|, p)$. This makes HE-Adaptive more resistant to distribution-aware attacks, but also enables $\mathcal{A}$ to explicitly know d-grams in a decoded vault. For the first thing, preliminary results show that when d=5, the average ranks of distribution-aware attacks in [19, 25, 29] against HE-Adaptive are 28.60%∼45.01%, forcing $\mathcal{A}$ to do 1.03∼1.10 times online verifications compared with adaptive PPMs boosting 5-grams based on $B(|\Theta|-|v|, 20\%)$. For the second thing, these d-grams are from decoy vaults, so $\mathcal{A}$ cannot use them to identify the real vault. In general, our HE-Adaptive makes honey vaults more resistant without bringing additional risks.

5 Security and Performance Evaluations

5.1 Security evaluations

Experimental setups. We use all existing honey vault (HV) distinguishing attacks in Table 1 to evaluate the security of our VaultGuard-NLE with HE-Adaptive. To know how it enhances HV security over existing schemes, we also experiment on Golla-NLE [29], Cheng-NLE [20] with 5-gram UPPM and PCFG-based [72] NoCrack-NLE [17]. Particularly, for adaptive Golla-NLE, we follow the same process in [29] to select 5-grams, and other setups are the same as Sec. 4.1. Besides, to reveal the security enhancement of HE-Adaptive, we implement it with VaultGuard-NLE, Golla-NLE [29] and Cheng-NLE [20], respectively; We set the boost parameter k_C=2 (following the result in Sec. 3.4) for selected 5-rvgrams. Our results are obtained under n=1,000 as in [17, 19, 20, 25, 29, 50].

Experimental results. Fig. 6 shows the CDGs of HV schemes (i.e., NLEs) against all eight existing HV distinguishing attacks. Overall, *our proposed VaultGuard-NLE with HE-Adaptive outperforms others, since the corresponding curve (green) is the closest to the ideal baseline against all attacks.* More precisely, the average rank $\bar{r}$ of our scheme is 41.15%∼53.94%, while for Golla-NLE [29], Cheng-NLE [19] and NoCrack-NLE [17], it ranges from 29.34%∼47.71%, 28.54%∼52.95% and 12.31%∼45.87%, respectively. This means that our VaultGuard-NLE with HE-Adaptive can force $\mathcal{A}$ to perform 1.10∼3.98 times online verifications than the state-of-the-art NLEs. Here, the "times online verification" is calculated as the ratio of average ranks of one HV scheme to another. For instance, the number of online verifications of VaultGuard-NLE with HE-Adaptive to adaptive Golla-NLE [29] is 1.10=45.01%/40.86% when both HV schemes are attacked by the theoretically-grounded attack [25]. *Thus, our VaultGuard can well resist HV distinguishing attacks.*

Table 2 shows the average rank of sequence-based VaultGuard-NLE, Golla-NLE [29] and Cheng-NLE [19] with and without HE-Adaptive in resisting these attacks. Overall, the values with HE-Adaptive are indeed closer to the ideal 50%, and HE-Adaptive alone forces $\mathcal{A}$ to perform 1.02∼1.71 times online verifications.

Our evaluation results corroborate the effectiveness of our VaultGuard-NLE with HE-Adaptive. First, as revealed in Sec. 4.1, our more accurate RPPM makes decoy vaults more dependent on the real vault than Golla-NLE [29] and Cheng-NLE [19]. Second, our HE-Adaptive achieves its design goal and can well mitigate the risk from $\mathcal{A}$'s accessibility of the adaptive PPM [29]. In summary, our VaultGuard scheme can effectively resist various honey vault distinguishing attacks by satisfying Criteria #1∼#3 in Table 1.

The results obtained under n=1,000 can be applied to other n [17, 29]. To substantiate this, we further compute the average ranks of VaultGuard under $n \in \{100, 500, 5000, 10^4\}$, and find the maximum

Table 2: The security against different distinguishing attacks against VaultGuard with and without HE-Adaptive.[†]

Honey vault distinguishing attack	Our VaultGuard-NLE			Adaptive Golla-NLE [29]			Static Cheng-NLE [20]		
	HE-Adaptive	No HE-Adaptive	Times online verifications	HE-Adaptive	No HE-Adaptive	Times online verifications	HE-Adaptive	No HE-Adaptive	Times online verifications
KL divergence attack [29]	50.53%	46.35%	**1.09**	41.28%	39.81%	**1.04**	50.60%	45.41%	**1.11**
Single password attack [19]	47.93%	29.94%	**1.60**	41.60%	40.55%	1.03	48.73%	28.54%	**1.71**
Password similarity attack [19]	48.88%	41.01%	**1.19**	28.60%	29.34%	0.97	43.09%	42.55%	**1.01**
Theoretically-grounded attack [25]	41.05%	35.24%	**1.16**	45.01%	40.86%	**1.10**	43.84%	31.68%	**1.38**
Weak encoding attack [20]	53.94%	52.95%	**1.02**	49.59%	47.71%	**1.04**	54.21%	52.95%	**1.02**
Strong encoding attack [20]	49.30%	49.17%	1.00	49.62%	43.26%	**1.15**	47.86%	49.17%	0.97
Adaptive extra attack [19]	48.38%	–	–	50.95%	37.42%	**1.36**	47.73%	–	–
(Adaptive) hybrid attack [19]	49.84%	30.12%	**1.66**	48.77%	34.82%	**1.40**	48.73%	29.09%	**1.68**

† The result is measured in average rank $\bar{r}$. The closer to 50%, the more secure the scheme is. The adaptive Golla-NLE [29] without HE-Adaptive is the one that takes k_C=5 and p=20% as in [29]. Since Cheng-NLE [20] includes no adaptive mechanism, we only compare the static version. For each NLE, our HE-Adaptive can better resist distribution-aware attacks without bringing vulnerabilities to adaptive extra and hybrid attacks [19]. The "times online verification" column is calculated as the ratio of the average ranks of the HV scheme with HE-Adaptive to that without, e.g., 1.09=50.53%/46.35%.

difference ranges from 0.35%~3.20% (see Table 6 in Appendix D). Hence, as claimed in [17, 29], larger n settings are unnecessary.

Remark 5. Our evaluations above can also be applied to Rao et al.'s master password guessing attack (MPGA) that exploits users' master password reset behaviors. They assumed that $\mathcal{A}$ can crack both old and new master passwords in n attempts, and claimed that MPGA can reach a near 100% success rate [50]. *We reveal that MPGA is either trivial or equivalent to ordinary HV distinguishing attacks*: (1) Since decoy vaults decrypted with old and new master passwords are mainly different, when the real vault is in both new and old vault lists, the real vault can be undoubtedly identified with a near 100% success rate; (2) For a more general case where only one master password is correctly guessed, MPGA is naturally reduced to the ordinary 1 out of $2{\cdot}n$ HV distinguishing attacks. As a mitigation, Rao et al.'s SMART scheme [50] maps a user's old and new master passwords to the same cognate master password, so a decoy vault decrypted with old and new master passwords is identical. In this case, security evaluation results against existing NLEs can also be readily applied to NLEs mounted with SMART.

5.2 Performance evaluations

Implementation setups. To more realistically evaluate the performance, we provide a proof-of-concept implementation of our VaultGuard as a password manager extension installed on the browser. We realize VaultGuard-NLE with HE-Adaptive using JavaScript ES2023, and Web Crypto API (see https://bit.ly/4m8UkRo) for cryptographic executions. Similarly to [17, 19, 29], we use AES in CTR mode with PBKDF2-SHA256 to execute symmetric password-based encryption (PBE). Our implementation supports functionalities commonly found in ordinary password vaults, including registration, password modification, account deletion, and login.

To mitigate users' master password typos (which may occur at a rate of 4.65%~8.06% [16, 18]), we use color hints as suggested in [12, 17, 19]. In honey vault registration, a color (e.g., green, blue, yellow) depending on the user's master password hash appears, and is deleted after registration. If the user finds the color different the next time she logs in, she might know that the entered master password is incorrect (see Fig. 7 in Appendix D). Since $\mathcal{A}$ does not know the user's color in advance, $\mathcal{A}$ gains no extra advantages in distinguishing attacks. Nevertheless, the effectiveness of color hints has not been substantiated. We will discuss them in Sec. 6.2.

Since a user has 168 accounts on average [59], we generate a vault with 200 passwords as the user's real vault. To simulate client-side usages, we do experiments on an ordinary laptop with Intel Core i9-14900HX 2.2 GHz CPU and 32GB memory. The average time and storage costs of 1,000 executions are shown in Table 3.

Table 3: Computation and storage costs of our VaultGuard.

Code length	128 bits	192 bits	256 bits
Encode a password	0.26 ms	0.33 ms	0.38 ms
Decode a password	0.01 ms	0.01 ms	0.01 ms
Encode a vault[†]	47.8 ms	65.8 ms	79.2 ms
Decode a vault[†]	1.20 ms	1.21 ms	1.23 ms
Encrypt a vault[†]	61.4 ms	76.4 ms	87.1 ms
Recover a vault[†]	9.00 ms	10.1 ms	11.7 ms
Add a password	10.6 ms	11.8 ms	13.9 ms
Recover a password[‡]	0.12 ms	0.12 ms	0.13 ms
Encrypted vault file	1.03 MB	1.54 MB	2.04 MB
Overall storage cost[§]	15.7 MB	16.0 MB	16.6 MB

† This process involves encoding/decoding and encoding-then-encrypting /decrypting-then-decoding a vault containing 200 passwords.

‡ This process includes decryption, in which we only need to decrypt the code of the passwords we need instead of the code of the entire vault.

§ A honey vault scheme requires locally storing the password probability model (PPM) regardless of whether HE-Adaptive is implemented, so the overall storage cost is the combination of the vault and PPM files.

Performance results show that VaultGuard is computationally efficient in different code lengths (128, 192, and 256 bits). Encrypting and recovering the entire vault only need 61.4ms~87.1ms and 9.00ms~11.7ms, respectively, ensuring unnoticeable latency for users. The encrypted vault file size scales with code length from 1.03MB to 2.04MB, and the total storage cost along with PPM is 15.7MB~16.6MB, which is acceptable compared with off-the-shelf password managers (see Table 5 in Appendix D). In all, our VaultGard (with HE-Adaptive) is efficient for practical deployment.

6 Discussions and Limitations

6.1 Security impacts

Impacts of evaluation results. Since the average rank $\bar{r}$ measuring honey vault security is quite stable across n [17, 29] (also confirmed in VaultGuard, see Table 6 in Appendix D), the absolute number $\bar{g}$ of the attacker $\mathcal{A}$'s online guesses/verifications can be calculated as $\bar{g}{\approx}\bar{r}{\cdot}n$ (see Sec. 2.3) when n depending on the master password guessability is estimated. This also enables estimating the time cost of $\mathcal{A}$'s online guesses with real-world login policies.

We use a concrete example to show this. Suppose that the user's master password is of medium strength and can resist 10^8 offline guesses (stronger than 50% of passwords, see Fig. 6 of [78]). Compared with existing honey vault schemes in [17, 19, 29], the "1.10~3.98 times" improvements in VaultGuard force $\mathcal{A}$ to do 10^7~$2.98{\times}10^8$ *more* online guesses/verifications. Particularly, if $\mathcal{A}$ online verifies on one website that forbids automatic logins (adopted

by 51% of websites in [9]) and enforces a stringent login policy allowing 100∼1,440 consecutive failed logins per day as adopted by high-profile services [65] and recommended by NIST SP 800-63B [32], $\mathcal{A}$ needs $1.67\times10^5\sim7.15\times10^7$ *more* hours on manual logins. Even if $\mathcal{A}$ distributes online guesses across 200 websites (as assumed in Sec. 5.2) deployed with the same/similar rate-limitings, she has to spend $833\sim3.58\times10^5$ *more* hours, and try over 2.05×10^5 online guesses per account (when using the most effective theoretically-grounded attack [25], see Table 2). Such a number of login attempts can be thwarted by defenses like rate-limiting and lockout.

We now discuss ways to enhance honey vault security in practice. First, a honey vault integrated with a password strength meter can translate the average rank into online verification numbers and time costs to nudge users towards strong master passwords. For instance, VaultGuard can inform the user whose master password is "`password12`" (resisting about 4,000 offline guesses, estimated by Zxcvbn [73]) that *her vault can be cracked in two weeks*. This is likely to be effective because users are likely to create a strong master password due to the importance of password vaults [55, 66], and various methods can be used to remember (e.g., automated memorable passwords [5, 7]) and retrieve (e.g., recovery code [1, 8]) strong master passwords. Second, distributed attacks can be mitigated by placing a number of decoy accounts in the vault [20, 68]: Once a decoy account is attempted, an alarm (e.g., via automatic emails) can be raised by the corresponding websites to nudge the user to change her master password and passwords in the vault.

Mitigation when rate-limiting fails. As shown above, the practical effectiveness of a honey vault relies on the *universal* online rate-limiting deployment. However, in practice, $\mathcal{A}$ may be able to identify (e.g., by using the automatic tool in [9]) websites allowing a large or even unlimited number of failed logins (termed as unprotected websites), and verify online without being detected/thwarted.

We note that all passwords in the vault, including those of unprotected websites, should be honey-encrypted. In this way, a honey vault is at least more secure than conventional vaults, because $\mathcal{A}$ has to online verify and wait for feedback: Even if $\mathcal{A}$ can online verify as fast as one login per second without being locked out, her time cost is still more significant than offline master password guessing attacks allowing $10^9\sim10^{12}$ guesses per day [23]. In contrast, storing plaintext passwords for unprotected websites can indeed isolate the impacts of unlimited guesses, but the corresponding passwords are left in plaintext and less secure than conventional vaults.

To further reduce $\mathcal{A}$'s guessing efficiency on unprotected websites, inspired by Chatterjee et al.'s HE-DH2 [17], we may store URLs of unprotected websites in salted guessing-resistant hashes (e.g., memory-hard Argon2i [11]). In this way, if $\mathcal{A}$ online guesses on an unprotected website, she must first recover its plaintext URL, which requires expensive offline guessing attacks. By careful design, the time cost can be acceptable for users but prohibitive for attackers trying large numbers of guesses. Experimental results on the same laptop CPU in Sec. 5.2 show that when VaultGuard employs URL hashes with Argon2i of 8MB memory usage, the user needs about 702 ms in hashing URLs of 200 websites. In contrast, $\mathcal{A}$ needs 97 hours for 10^8 offline guesses against a single URL hash with the same CPU, and even more hours with GPU/FPGA/ASIC, because "implementing it (Argon2i) on dedicated cracking hardware should be neither cheaper nor faster (than the x86 architecture)" [11].

Limitation on the vault dataset. Since `Pastebin` is the *only* available password vault dataset and its size is relatively small (see Sec. 2.4), using it may cause unwanted bias. Hence, more diverse vault datasets are needed. Unfortunately, there is no other public data for leaked password vaults (e.g., LastPass [52, 58, 74] and Opera [45]). To address this issue, one alternative is matching a user's multiple cross-site passwords through the email address as in [46, 67, 69, 75, 77]). Nevertheless, whether they can represent a user's passwords stored in the real vault remains an open question.

Limitation on PPMs. In practice, passwords from different languages and services are distributed differently [66], so a future improvement is to group websites of the same language and similar services into the same category and design service-specific NLEs. Additionally, $\mathcal{A}$ may use various personally identifiable information (PII, e.g., birthdays and names [66, 67]) to facilitate distinguishing attacks. Thus, it is crucial to design user-specific PII-assisted PPMs, particularly to make TarGuess-I [67], RFGuess-PII [70], and PassBert-PII [77] models ambiguity-free for building NLEs. To better resist the policy attack [19, 29], a promising solution is to encode string types (e.g., letters, digits, and symbols) and password lengths before contents. We leave detailed designs for future research.

Limitation on resisting password leakage attacks. A drawback of honey vaults is that $\mathcal{A}$ who knows one or more leaked passwords in the vault can directly identify the real vault, as the one with the leaked passwords is real. This reduces the security of honey vaults to that of conventional vaults. Three possible measures exist, but each has its pros and cons. The first is to make the master password work with a user-possessed hardware (e.g., YubiKey [2]) on the client side. For instance, for each password in the vault, the hardware derives and stores a low-entropy random integer $z\in U[1, 10^3]$ to control the number h of SHA256 iterations in PBKDF2 master password hashes, e.g., $h=z\times10^4+10^3$. However, usability burdens can be incurred by the user-possessed hardware. Second, password-hardened encryption (PHE) [37] (see Sec. 1) deployed on the server side of a honey vault can inherently address threats of offline master password guessing and password leakage attacks, if the authentication server and the external crypto-server of the master password are not simultaneously compromised [37]. However, PHE can be expensive due to the crypto-server, and its failure can make PHE nonfunctional, causing a single point of failure. Third, a password leakage detection service (e.g., HIBP [6], see Table 5 in Appendix D) can be integrated, but it is only effective once the leaked dataset has been added to the service database. Thus, $\mathcal{A}$ who exploits newly leaked datasets cannot be thwarted.

6.2 Potential countermeasures for master password typo detection

Depending on whether a trusted authentication server can be deployed in conjunction with a honey vault, we discuss potential ways to identify master password typos. Here, a server is trusted if it cannot be compromised when the honey vault file is leaked.

Client-side ways require no trusted server and can be deployed for both local-only (where the vault file is only stored on the client side) and server-based (where the vault can be uploaded to the server in the cloud) honey vaults. Firstly, for color hints in [12, 17, 19], the rate at which the user unknowingly enters a master password typo (when the displayed color remains the same as the correct

one) becomes $1/b$ of the original rate if b colors are used, e.g., from 4.65%∼8.06% to 0.93%∼1.61% with b=5. Nevertheless, the rate is not zero. In this case, the user has to log in to websites to find the master password incorrect, log out, and re-enter her honey vault.

Alternatively, to mitigate typos, a honey vault can display passwords of frequently logged-in websites (e.g., institutional email) in a more explicit manner (e.g., in descending order of frequency and darkened color). When different master passwords are entered, *only* decrypted passwords are different, while accounts of frequently logged-in websites (which the user is most familiar with) are displayed in the same manner *independent of master passwords:* The differences between the correct and decrypted passwords of these websites can serve as reminders. In this way, *the user is likely to observe differences even if she may not memorize the exact correct passwords, and $\mathcal{A}$ cannot obtain extra advantages in offline ruling out decoy vaults.* For instance, if the user's most frequently logged-in website is `Gmail` and the associated password is "`I0veu4ever`" (e.g., in the top and in bold); When the user accidentally makes a typo in her master password, the decrypted password of `Gmail` will be different, so the typo can be noticed by the user.

Both measures bring memory burdens to users: Color hints require memorizing a random color of $\log_2 b$ (e.g., $\log_2 5$=2.32) bits, and the effectiveness of displaying frequently logged-in websites depends on whether the user is familiar with, at least parts of, her frequently logged-in passwords. Incorporating fewer colors and websites may ease memory burdens, but may also reduce the effectiveness of identifying typos. Further, both measures require users' manual inspection to identify differences: Users who are less aware of color and password changes may find such measures non-informative. These measures are also vulnerable to shoulder-surfing attacks: Color hints and displayed passwords may be observed by $\mathcal{A}$ either in a nearby position (e.g., standing behind) or employing a remote tool (e.g., camera) when the user logs in, which facilitates honey vault distinguishing attacks. Overall, there still lack user studies to quantify the effectiveness and the memory burdens of both measures, and we leave it as future research.

Server-side ways require an additional trusted server that expands the trust assumption in the security model (see Sec 2.3). First, the server deploys a typo-detection website that can either be an independent or an existing one allowing only limited consecutive failed logins (e.g., five per hour in Amazon [65]). Once the honey vault is decrypted, the decrypted password of the typo-detection website is automatically logged in. Within the lockout threshold, the user can know if an entered master password is correct. Compared with other sites, the typo-detection website only involves one more bit indicating whether the account/website is normal, and its leakage poses no security risks. This measure can be applied to both local-only and server-based honey vaults. Second, for a server-based honey vault that can securely store (e.g., on another domain) master password credentials, password-related cryptographic protocols (e.g., [10, 16, 18]) can be further employed to detect and correct master password typos. Note that the effectiveness of these ways relies on the trusted status of the additional server. Otherwise, after compromising the typo-detection website, $\mathcal{A}$ can: (1) log in with incorrect passwords to trigger false breach alarms on the honey vault; (2) offline recover the password hash stored on the typo-detection server; (3) trial-decrypt the honey vault until the decrypted password of the typo-detection site equals the recovered one in Step (2), indicating that the real vault is identified. Similarly, if the authentication server used for detecting typos is compromised, $\mathcal{A}$ can offline guess the master password from its stored credentials.

7 Conclusion

We have systematically tackled how to best generate (and evaluate) honey vaults, and proposed VaultGuard with HE-Adaptive scheme as the solution. We, *for the first time*, propose three design criteria for secure honey vault schemes, and each aims to address one type of vulnerability in existing critical natural language encoders (NLEs). Particularly, our VaultGuard-NLE is comprised of an accurate sequence-based password probability model and an efficient password reuse probability model, and our HE-Adaptive honey-encrypts the user's employed adaptive PPM from being known by attackers. Evaluation results reveal that our scheme significantly outperforms its counterparts in resisting various honey vault distinguishing attacks, and our proof-of-concept implementation is efficient. We also discuss a few practical security and usability issues when deploying honey vaults. Given the wide recommendation of password managers/vaults and their catastrophic impacts once compromised, we believe that our honey vault design criteria (and new proposals) will be of broad interest and push honey vault research towards mathematical rigor and practical deployment.

Acknowledgment

The authors are grateful to the shepherd and anonymous reviewers for their invaluable comments. Ding Wang is the corresponding author. This research was in part supported by the National Natural Science Foundation of China (NSFC) under Grants Nos. 62222208 and 62172240, and by the Fundamental Research Funds for the Central Universities, Nankai University (Grant No. 63253229). This is the full version of the ACM CCS 2025 paper, and the artifact is available at https://bit.ly/42blHTH .

References

[1] *Account recovery options for Dashlane*, Aug. 2024, https://support.dashlane.com/hc/en-us/articles/11282971791634-Account-recovery-options-for-Dashlane.
[2] *YubiKey 5 Series: The multi-protocol, highest assurance security key that enables passwordless*, 2024, https://www.yubico.com/products/yubikey-5-overview/.
[3] *Feds link $150M cyberheist to 2022 LastPass hacks*, Mar. 2025, https://krebsonsecurity.com/2025/03/feds-link-150m-cyberheist-to-2022-lastpass-hacks/.
[4] *FIDO2 passwordless authentication*, 2025, https://www.yubico.com/authentication-standards/fido2/.
[5] *Generating pronounceable passwords in Enpass*, 2025, https://support.enpass.io/app/generate/generating_pronounceable_random_passwords_in_enpass.htm.
[6] *Have I been pwned? Pwned passwords*, 2025, https://haveibeenpwned.com/Passwords.
[7] *Strong. Secure. Awesome. Try our random password generator*, 2025, https://1password.com/password-generator.
[8] *Using 1Password Generate and use recovery codes*, May 2025, https://support.1password.com/recovery-codes/?windows.
[9] S. Al Roomi and F. Li, "A large-scale measurement of website login policies," in *Proc. USENIX SEC 2023*, pp. 2061–2078.
[10] M. H. Ameri and J. Blocki, "Conditional encryption with applications to secure personalized password typo correction," in *Proc. ACM CCS 2024*, pp. 4643–4657.
[11] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in *Proc. IEEE EuroS&P 2016*, pp. 292–302.
[12] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, "Kamouflage: Loss-resistant password management," in *Proc. ESORICS 2024*, pp. 286–302.
[13] J. Bonneau, C. Herley, P. Van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Commun. ACM*, vol. 58, no. 7, pp. 78–87,

2015.
[14] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE S&P 2012*, pp. 553–567.
[15] J. Brost, C. Egger, R. W. Lai, F. Schmid, D. Schröder, and M. Zoppelt, "Threshold password-hardened encryption services," in *Proc. ACM CCS 2020*, pp. 409–424.
[16] R. Chatterjee, A. Athayle, D. Akhawe, A. Juels, and T. Ristenpart, "pASSWORD tYPOS and how to correct them securely," in *Proc. IEEE S&P 2016*, pp. 799–818.
[17] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart, "Cracking-resistant password vaults using natural language encoders," in *Proc. IEEE S&P 2015*, pp. 481–498.
[18] R. Chatterjee, J. Woodage, Y. Pnueli, A. Chowdhury, and T. Ristenpart, "The typtop system: Personalized typo-tolerant password checking," in *Proc. ACM CCS 2017*, pp. 329–346.
[19] H. Cheng, W. Li, P. Wang, C.-H. Chu, and K. Liang, "Incrementally updateable honey password vaults," in *Proc. USENIX SEC 2021*, pp. 857–874.
[20] H. Cheng, Z. Zheng, W. Li, P. Wang, and C. Chu, "Probability model transforming encoders against encoding attacks," in *Proc. USENIX SEC 2019*, pp. 1573–1590.
[21] C. Clifford and P. Sharon, *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/.
[22] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA: Wiley-Interscience, 2006.
[23] S. Croley, *RTX 5090 CUDA v6.2.6-851.Benchmark*, Feb. 2025, https://gist.github.com/Chick3nman/09bac0775e6393468c2925c1e1363d5c.
[24] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proc. NDSS 2014*, pp. 1–15.
[25] F. Duan, D. Wang, and C. Jia, "A security analysis of honey vaults," in *Proc. IEEE S&P 2024*, pp. 1424–1442.
[26] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.
[27] X. Gao, Y. Yang, C. Liu, C. Mitropoulos, J. Lindqvist, and A. Oulasvirta, "Forgetting of passwords: Ecological theory and data," in *Proc. USENIX SEC 2018*, pp. 221–238.
[28] A. Gavazzi, R. Williams, E. Kirda, L. Lu, A. King, A. Davis, and T. Leek, "A study of Multi-Factor and Risk-Based authentication availability," in *Proc. USENIX SEC 2023*, pp. 2043–2060.
[29] M. Golla, B. Beuscher, and M. Dürmuth, "On the security of cracking-resistant password vaults," in *Proc. ACM CCS 2016*, pp. 1230–1241.
[30] M. Golla, G. Ho, M. Lohmus, M. Pulluri, and E. M. Redmiles, "Driving 2FA adoption at scale: Optimizing Two-Factor authentication notification design patterns," in *Proc. USENIX SEC 2021*, pp. 109–126.
[31] N. Goud, *UK says NO to ransom passwords such as admin, 123456 and qwerty*, April 2024, https://www.cybersecurity-insiders.com/uk-says-no-to-ransom-passwords-such-as-admin-123456-and-qwerty/.
[32] P. A. Grassi, E. M. Newton, R. A. Perlner, and et al., "NIST 800-63B digital identity guidelines: Authentication and lifecycle management," McLean, VA, Tech. Rep., Mar. 2020.
[33] C. Herley and S. E. Schechter, "Distinguishing attacks from legitimate authentication traffic at scale," in *Proc. NDSS 2019*, pp. 1–13.
[34] M. Islam, M. S. Bohuk, P. Chung, T. Ristenpart, and R. Chatterjee, "Araña: Discovering and characterizing password guessing attacks in practice," in *Proc. USENIX SEC 2023*, pp. 1019–1036.
[35] J. Jaeger, T. Ristenpart, and Q. Tang, "Honey encryption beyond message recovery security," in *Proc. EUROCRYPT 2016*, pp. 758–788.
[36] A. Juels and T. Ristenpart, "Honey encryption: Security beyond the brute-force bound," in *Proc. EUROCRYPT 2014*, pp. 293–310.
[37] R. W. Lai, C. Egger, M. Reinert, S. S. Chow, M. Maffei, and D. Schröder, "Simple password-hardened encryption services," in *Proc. USENIX SEC 2018*.
[38] R. Lakshmanan, *KeePass exploit allows attackers to recover master passwords from memory*, May 2023, https://thehackernews.com/2023/05/keepass-exploit-allows-attackers-to.html.
[39] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin, "A measurement study of authentication rate-limiting mechanisms of modern websites," in *Proc. ACSAC 2018*.
[40] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE S&P 2014*, pp. 689–704.
[41] W. Melicher, B. Ur, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean and accurate: Modeling password guessability using neural networks," in *Proc. USENIX SEC 2017*, pp. 175–191.
[42] C. W. Munyendo, P. Mayer, and A. J. Aviv, ""I just stopped using one and started using the other": Motivations, techniques, and challenges when switching password managers," in *Proc. ACM CCS 2023*, pp. 3123–3137.
[43] S. Oesch and S. Ruoti, "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.
[44] S. Oesch, S. Ruoti, J. Simmons, and A. Gautam, ""It basically started using me:" An observational study of password manager usage," in *Proc. ACM CHI 2022*.
[45] *Opera server breach incident*, Opera, Aug. 2016, https://blogs.opera.com/security/2016/08/opera-server-breach-incident/.
[46] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE S&P 2019*.
[47] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, "Improving password guessing via representation learning," in *Proc. IEEE S&P 2021*.
[48] P. Peng, C. Xu, L. Quinn, H. Hu, B. Viswanath, and G. Wang, "What happens after you leak your password: Understanding credential sharing on phishing sites," in *Proc. ACM ASIACCS 2019*, pp. 181–192.
[49] A. Peslyak, *John the Ripper password cracker*, May 2019, http://www.openwall.com/john/.
[50] T. Rao, Y. Su, P. Xu, Y. Zheng, W. Wang, and H. Jin, "You reset I attack! A master password guessing attack against honey password vaults," in *Proc. ESORICS 2023*.
[51] H. Shittu, *LastPass Data Breach Results in $4.4 Million Crypto Loss for 25 Victims in a Single Day*, Oct. 2023, https://cryptonews.com/news/lastpass-data-breach-results-in-4-4-million-crypto-loss-for-25-victims-in-a-single-day/.
[52] J. Siegrist, *LastPass hacked - Identified early and resolved*, June 2015, https://blog.lastpass.com/2015/06/lastpass-security-notice/.
[53] S. Stephenson, B. Pal, S. Fan, E. Fernandes, Y. Zhao, and R. Chatterjee, "Sok: Authentication in augmented and virtual reality," in *Proc. IEEE S&P 2022*.
[54] J. Steube, *Hashcat*, Aug. 2025, https://hashcat.net/hashcat/.
[55] E. Stobert and R. Biddle, "The password life cycle," *ACM Trans. Priv. Secur.*, vol. 21, no. 3, pp. 13:1–13:32, 2018.
[56] D. R. Thomas, S. Pastrana, A. Hutchings, R. Clayton, and A. R. Beresford, "Ethical issues in research using datasets of illicit origin," in *Proc. IMC 2017*, pp. 445–462.
[57] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki *et al.*, "Data breaches, phishing, or malware? Understanding the risks of stolen credentials," in *Proc. ACM CCS 2017*.
[58] K. Toubba, *Notice of Recent Security Incident*, Dec. 2022, https://blog.lastpass.com/posts/2022/12/notice-of-recent-security-incident.
[59] K. Viezelyte, *Juggling security: How many passwords does the average person have in 2024?*, April 2024, https://nordpass.com/blog/how-many-passwords-does-average-person-have/.
[60] A. Vigderman, *America's Password Habits*, Nov. 2024, https://www.security.org/resources/online-password-strategies/.
[61] H. Von Stackelberg, *Market structure and equilibrium*. Springer Science & Business Media, 2010.
[62] Z. Whittaker, *Why you need to use a password manager*, Dec. 2018, https://techcrunch.com/2018/12/25/cybersecurity-101-guide-password-manager/?guccounter=1.
[63] C. Wang, S. T. Jan, H. Hu, D. Bossart, and G. Wang, "The next domino to fall: Empirical analysis of user passwords across online services," in *Proc. CODASPY 2018*, pp. 196–203.
[64] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 11, pp. 2776–2791, 2017.
[65] D. Wang, X. Shan, Q. Dong, Y. Shen, and C. Jia, "No single silver bullet: Measuring the accuracy of password strength meters," in *Proc. USENIX SEC 2023*, pp. 947–964.
[66] D. Wang, P. Wang, D. He, and Y. Tian, "Birthday, name and bifacial-security: Understanding passwords of Chinese web users," in *Proc. USENIX SEC 2019*.
[67] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM CCS 2016*, pp. 1242–1254.
[68] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang, "How to attack and generate honeywords," in *Proc. IEEE S&P 2022*, pp. 966–983.
[69] D. Wang, Y. Zou, Y.-A. Xiao, S. Ma, and X. Chen, "PASS2EDIT: A multi-step generative model for guessing edited passwords," in *Proc. USENIX SEC*, 2023.
[70] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. USENIX SEC 2023*, pp. 965–982.
[71] L. Wang, Y. Li, and K. Sun, "Amnesia: A bilateral generative password manager," in *Proc. ICDCS 2016*, pp. 313–322.
[72] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. IEEE S&P 2009*, pp. 391–405.
[73] D. Wheeler, "Zxcvbn: Low-budget password strength estimation," in *Proc. USENIX SEC 2016*, pp. 157–173.
[74] L. Whitney, *LastPass CEO reveals details on security breach*, May 2011, https://www.cnet.com/news/privacy/lastpass-ceo-reveals-details-on-security-breach/.
[75] K. Xiu and D. Wang, "PointerGuess: Targeted password guessing model using pointer mechanism," in *Proc. USENIX SEC 2024*, pp. 5555–5572.
[76] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han, "Chunk-level password guessing: Towards modeling refined password composition representations," in *Proc. ACM CCS 2021*, pp. 5–20.
[77] M. Xu, J. Yu, X. Zhang, C. Wang, S. Zhang, H. Wu, and W. Han, "Improving real-world password guessing attacks via bi-directional Transformers," in *Proc. USENIX SEC 2023*, pp. 1001–1018.
[78] T. Yang and D. Wang, "RankGuess: Password guessing using adversarial ranking," in *Proc. IEEE S&P 2025*, pp. 682–700.
[79] S. Zibaei, D. R. Malapaya, B. Mercier, A. Salehi-Abari, and J. Thorpe, "Do password managers nudge secure (random) passwords?" in *Proc. SOUPS 2022*, pp. 581–597.
[80] V. Zimmermann, "From the quest to replace passwords towards supporting secure and usable password creation," Ph.D. dissertation, 2021.

A Other Distinguishing Attacks

KL divergence attack [29] exploits the KL divergence between the password distributions of the real and decoded decoy vaults. The larger the KL divergence, the more discrepancy between the real and decoy vaults. Formally, the score function is

$$\mathrm{s_{KL}}(v) = \sum_{pw\in v} \mathrm{Pr_f}(pw)\cdot\log\Big(\frac{\mathrm{Pr_f}(pw)}{\mathrm{Pr_D}(pw)}\Big), \tag{10}$$

where $\mathrm{Pr_f}(pw)$ is the probability of a password in the vault, i.e., $pw{\in}v$, and $\mathrm{Pr_D}(pw)$ is the probability of pw decoy vaults induced by the NLE. If pw does not exist in any decoy vault, there is $\mathrm{s_{KL}}(v){=}1$.

Single password attack [19]. Cheng et al. [19] proposed the single password attack based on the maximum likelihood heuristics. Particularly, they assumed that passwords in each vault are independent, and obtain the score function as

$$\mathrm{s_{SP}}(v) = \prod_{pw\in v} f_{\mathrm{SP}}(pw), \tag{11}$$

where $f_{\mathrm{SP}}(pw)$ is the smoothed probability given as follows:

$$f_{\mathrm{SP}}(pw) = \begin{cases} 1, & \text{If } \mathtt{Count}(pw) < 5 \text{ and } \frac{f_a^*(pw)}{\mathrm{Pr_{DS}}(pw)} > 1 \\ \frac{f_a^*(pw)}{\mathrm{Pr_D}(pw)}, & \text{Otherwise.} \end{cases} \tag{12}$$

In Eq. 12, $f_a^*(pw)$ is the probability of pw in a model trained on the dataset D (e.g., Rockyou), i.e., $f_a^*(pw){=}\frac{\mathtt{Count}(pw)+1}{|\mathrm{D}|+1}$, and $\mathrm{Pr_D}(v)$ is the decoy probability sampled from decoding random codes.

Password similarity attack [19]. Due to reused passwords in the user's vault, Cheng et al. [19] also proposed the password similarity attack by exploiting the difference between similarity features in real and decoy vaults, and the score function is:

$$\mathrm{s_{PS}}(v) = \prod_{vf\in\mathcal{VF}} \frac{\phi_1(v,vf)}{\phi_2(v,vf)}, \tag{13}$$

where vf denotes the feature, $\mathcal{VF}$ is the feature set, and $\phi_1(v,vf)$ (resp. $\phi_2(v,vf)$) is the probability of the real (resp. decoy) vault v having the feature vf, which can be defined as follows

$$\phi_1(v,vf) = \begin{cases} \frac{|\{v\in\mathcal{V}|\mathsf{VF}(v,vf)=1\}|}{|\mathcal{V}|} & \text{if } \mathsf{VF}(v,vf)=1 \\ 1 & \text{if } \mathsf{VF}(v,vf)=0, \end{cases} \tag{14}$$

where $\mathsf{VF}(v,vf)$ checks whether a vault v has the feature vf. If v has vf, $\mathsf{VF}(v,vf){=}1$, otherwise, $\mathsf{VF}(v,vf){=}0$.

Adaptive extra attack [19]. To attack Golla et al.'s adaptive mechanism [29] (see Sec. 2.2), Cheng et al. [19] exploited the relation between the real vault (which does not exist for decoy vaults) and the adaptive PPM, and proposed the adaptive extra attack ("extra" means extraction). They calculate the real-to-decoy ratio on the number of d-grams whose probabilities have been increased by the adaptive mechanism [19], and the score function is:

$$\mathrm{s_{AE}}(v) = \prod_{j=1}^{n^*}\Big(\frac{1}{p}\Big)^{m_j^*}\prod_{t=k_j+1}^{l_j}\frac{t-m_j^*}{m_j^*}, \tag{15}$$

where n^* is the number of unique passwords, m_j^* is the number of pw_j in v, l_j is the length of pw_j, k_j is the number of boosted d-grams in pw_j, and p is the probability that a d-gram is boosted.

(Adaptive) hybrid attack [19]. For static NLEs (i.e., without the adaptive mechanism [19]), the hybrid attack combines the single password attack [19] with the password similarity attack [19]. For adaptive NLEs, the adaptive hybrid attack further combines adaptive extra attack [19]. The score functions are given as $\mathrm{s_H}(v) = \mathrm{s_{SP}}(v)\cdot\mathrm{s_{PS}}(v)$ and $\mathrm{s_{AH}}(v) = |\mathrm{s_{SP}}(v)\cdot\mathrm{s_{PS}}(v)|$, respectively.

B Detailed Proofs

B.1 Proofs of Criterion #1

Lemma 1. *(χ^2 **divergence inequality [22]**). Let μ and ν be the probability mass functions of two distributions on a finite space Ω, then there is $\Delta_{\mathrm{KL}}(\mu,\nu){\le}\chi^2(\mu,\nu)$ between KL and χ^2 divergences.*

Proof. Since $\log(x)$ is concave, Jensen's inequality tells

$$\Delta_{\mathrm{KL}}(\mu,\nu){=}\sum_{x\in\Omega}\mu(x)\log\Big(\frac{\mu(x)}{\nu(x)}\Big) \le \log\Big(\sum_{x\in\Omega}\frac{\mu(x)^2}{\nu(x)}\Big). \tag{16}$$

Meanwhile, since $x-1{\ge}\log(x)$ for $x{\ge}1$, there is

$$\begin{aligned}\chi^2(\mu,\nu) &= \sum_{x\in\Omega}\frac{(\mu(x)-\nu(x))^2}{\nu(x)} = \sum_{x\in\Omega}\frac{\mu(x)^2}{\nu(x)} - 1 \\ &\ge \log\Big(\sum_{x\in\Omega}\frac{\mu(x)^2}{\nu(x)}\Big) \ge \Delta_{\mathrm{KL}}(\mu,\nu).\end{aligned} \tag{17}$$

□

Theorem 1. *For any user that has one real and arbitrary $n{-}1$ (with $n{<}|V|$, where $\mathcal{V}$ is the vault space) decoy vaults, when faced with Duan et al.'s theoretically-grounded distinguishing attack [25], the NLE can reduce this attack to uniform random guesses, i.e., making $\mathrm{s_{TG}}(v){=}\frac{\mathrm{Pr_V}(v_i)^2}{\mathrm{Pr_D}(v_i)}{\equiv}C$ constant for all $v_i{\in}\{v_1\ldots,v_n\}$ if and only if the normalized $\mathrm{Pr_D}(v_i){=}\mathrm{Pr_V}(v_i){=}\frac{1}{n}$ holds. Otherwise, the best an NLE can do is forcing $\mathcal{A}$ to guess in the descending order of $\mathrm{Pr_V}(v)$.*

Proof. When $\mathrm{Pr_V}(v_i){=}\mathrm{Pr_D}(v_i){=}\frac{1}{n}$ holds for any vault v_i, it is obvious that $\mathrm{s_{TG}}(v)$ is a constant regardless of v, so we focus on proving the other side. Following the Stackelberg game [61], the leader NLE first sets $\mathrm{Pr_D}(v)$ based on its estimated/approximated $\mathrm{Pr_V}(v)$, and the follower $\mathcal{A}$ devises her optimal guessing strategy. Thus, we use the orthodox backward induction to find the NLE's best strategy, i.e., how to set $\mathrm{Pr_D}(v)$. We first consider the follower $\mathcal{A}$'s optimal strategy. According to [25], $\mathcal{A}$ guesses in the descending order of the following Bayesian posterior probability:

$$C_0\frac{\mathrm{Pr_V}(v)}{\mathrm{Pr_D}(v)}\sum_{s\in\mathcal{S}}\sum_{mp\in\mathcal{MP}}\mathrm{Pr_{MP}}(mp)\Pr(\mathrm{enc}(mp,s)=c) = C_0\frac{\mathrm{Pr_V}(v)^2}{\mathrm{Pr_D}(v)}, \tag{18}$$

where $\mathrm{Pr_V}(v)$, $\mathrm{Pr_D}(v)$, $\mathrm{Pr_{MP}}(\cdot)$ $\mathrm{Pr_C}(\cdot)$ are normalized probabilities over the given n vaults. The equivalence holds since the assumptions that $\mathrm{Pr_{MP}}(mp)\cdot\Pr(\mathrm{enc}(mp,s){=}c){\approx}\mathrm{Pr_{MP}}(mp)$ and $\mathrm{Pr_{MP}}(mp){\approx}\mathrm{Pr_V}(v)$ (see Sec. 3.2). Besides, $C_0{=}\frac{1}{|\mathcal{S}|\mathrm{Pr_C}(c)}$ [25] equals 1 because $|\mathcal{S}|{=}n$ and the ciphertext of each vault, if it is real, is chosen with the probability $1/n$. Hence, we can have $\sum_{i=1}^n\frac{\mathrm{Pr_V}(v_i)^2}{\mathrm{Pr_D}(v_i)}{=}1$ for normalized $\mathrm{Pr_V}(v)$ and $\mathrm{Pr_D}(v)$. Based on these premises, we have the following constraints for the NLE with $\mathrm{Pr_D}(v)$ being the variables:

$$\begin{aligned}&\sum_{i=1}^n\frac{\mathrm{Pr_V}(v_i)^2}{\mathrm{Pr_D}(v_i)} = 1, \quad \sum_{i=1}^n\mathrm{Pr_D}(v_i) = 1, \sum_{i=1}^n\mathrm{Pr_V}(v_i) = 1 \\ &0 < \mathrm{Pr_D}(v_i) < 1, \quad 0 < \mathrm{Pr_V}(v_i) < 1 \quad \text{for all } i\in\{1,\ldots,n\}.\end{aligned} \tag{19}$$

In this way, we have the following by Cauchy-Schwarz inequality:

$$\begin{aligned}&\Big(\sum_{i=1}^n\frac{\mathrm{Pr_V}(v_i)}{\sqrt{\mathrm{Pr_D}(v_i)}}\sqrt{\mathrm{Pr_D}(v_i)}\Big)^2 \leqslant \Big(\sum_{i=1}^n\Big(\frac{\mathrm{Pr_V}(v_i)}{\sqrt{\mathrm{Pr_D}(v_i)}}\Big)^2\Big)\Big(\sum_{i=1}^n\Big(\sqrt{\mathrm{Pr_D}(v_i)}\Big)^2\Big) \\ &\text{i.e.,}\quad \Big(\sum_{i=1}^n\mathrm{Pr_V}(v_i)\Big)^2 \leqslant \Big(\sum_{i=1}^n\frac{\mathrm{Pr_V}(v_i)^2}{\mathrm{Pr_D}(v_i)}\Big)\Big(\sum_{i=1}^n\mathrm{Pr_D}(v_i)\Big).\end{aligned} \tag{20}$$

The equality holds if and only if $\frac{\mathrm{Pr_V}(v_i)}{\sqrt{\mathrm{Pr_D}(v_i)}}{=}\sqrt{\mathrm{Pr_D}(v_i)}$, i.e., $\mathrm{Pr_D}(v_i){=}\mathrm{Pr_V}(v_i)$ for all $i{\in}\{1,\ldots,n\}$. Otherwise, if there exists some $\mathrm{Pr_D}(v'){\neq}\mathrm{Pr_V}(v')$, there is $\big(\sum_{i=1}^n\frac{\mathrm{Pr_V}(v_i)^2}{\mathrm{Pr_D}(v_i)}\big)\big(\sum_{i=1}^n\mathrm{Pr_D}(v_i)\big){>}\big(\sum_{i=1}^n\mathrm{Pr_V}(v_i)\big)^2$

i.e., 1>1, leading to a contradiction. Hence, the only strategy for the NLE is setting $\mathrm{Pr_D}(v)=\mathrm{Pr_V}(v)$ for all $v\in\mathcal{V}$. This ends the proof. □

Corollary 2. *An NLE can resist the theoretically-grounded attack instantiated under* $s_{TG}(v)=\frac{\mathrm{Pr_V}(v)^2}{\mathrm{Pr_D}(v)}$ *(resp.* $s_2(v)=\frac{\mathrm{Pr_V}(v)^{\frac{n+1}{n}}}{\mathrm{Pr_D}(v)}$*) can also resist that instantiated under* $s_2(v)$ *(resp.* $s_{TG}(v)$*) by forcing* $\mathcal{A}$ *to guess vaults in the same order: For any two vaults* $v_i, v_j\in\{v_1, v_2, \ldots, v_n\}$, $s_2(v_i)\geq s_2(v_j)$ *if and only if* $s_{TG}(v_i)\geq s_{TG}(v_j)$.

Proof. For simplicity, we set $v\in\{v_1,\ldots,v_n\}$ for some $n>1$. To instantiate $s_2(v_i)=\frac{\mathrm{Pr_V}(v_i)^{\frac{n+1}{n}}}{\mathrm{Pr_D}(v_i)}$ [25], we first consider $\mathrm{Pr_{MP}}(mp) = \mathrm{Pr_D}(v)^{\frac{1}{n}}$ claimed in [25], and there should be $\sum_{i=1}^{n}\mathrm{Pr_{MP}}(mp_i) = \sum_{i=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{1}{n}}=1$. However, since $0<\mathrm{Pr_V}(v_i)<1$, $\sum_{i=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{1}{n}} > \sum_{i=1}^{n}\mathrm{Pr_V}(v)=1$, which leads to a contradiction. Thus, we mainly consider $\mathrm{Pr_{MP}}(mp)\propto\mathrm{Pr_V}(v)^{\frac{1}{n}}$, i.e., $\mathrm{Pr_{MP}}(mp_i) = \frac{\mathrm{Pr_V}(v_i)^{\frac{1}{n}}}{\sum_{j=1}^{n}\mathrm{Pr_V}(v_j)^{\frac{1}{n}}}$.

To begin with, like $s_{TG}(\cdot)$, we prove that $s_2(\cdot)$ cannot be constant for any $\mathrm{Pr_D}(v)$ if and only if all vaults are uniformly distributed, i.e., $\mathrm{Pr_V}(v)=\mathrm{Pr_D}(v)=1/n$. We mainly focus on proving the other side since $\mathrm{Pr_V}(v)=\mathrm{Pr_D}(v)=1/n$ can obviously make $s_2(v)$ constant. We use the proof by contradiction: we suppose that the solution $\mathrm{Pr_D}(v_i)=\frac{\mathrm{Pr_V}(v_i)^{\frac{n+1}{n}}}{\sum_{j=1}^{n}\mathrm{Pr_V}(v_j)^{\frac{n+1}{n}}}$ can make $s_2(\cdot)$ constant. Then, based on the left-hand side of Eq. 18 from [25], we can obtain

$$\sum_{i=1}^{n}\mathrm{Pr_{MP}}(mp_i)\frac{\mathrm{Pr_V}(v_i)}{\mathrm{Pr_D}(v_i)} = \left(\sum_{i=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{n+1}{n}}\right)\cdot n = C' = \sum_{i=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{1}{n}}, \quad (21)$$

where the second and third equalities come from the relations that there is $\sum_{i=1}^{n}\frac{\mathrm{Pr_V}(v_i)^{\frac{n+1}{n}}}{\mathrm{Pr_D}(v_i)C'}=1$ for normalized $\mathrm{Pr_V}(v)$ and $\mathrm{Pr_D}(v)$. According to Hölder's inequality that $(\sum_{i=1}^{n}|x_i|^{r+t})^r(\sum_{i=1}^{n}|y_i|^{r+t})^t \geq (\sum_{i=1}^{n}|x_i|^r|y_i|^t)^{r+t}$ for any $r,t\in\mathbb{R}^+$, we let $x_i=\mathrm{Pr_V}(v_i)$, $y_i=1$, $r=1/n$ and $t=1$, and have the following relationship:

$$\left(\sum_{i=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{n+1}{n}}\right)\cdot n \geq \left(\sum_{i=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{1}{n}}\right)^n > \sum_{i=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{1}{n}}, \quad (22)$$

which contradicts with Eq. 21. Therefore, $s_2(v)=\frac{\mathrm{Pr_V}(v)^{\frac{n+1}{n}}}{\mathrm{Pr_D}(v)}$ cannot be constant unless all real and decoy vaults are uniformly distributed.

We now prove the order-preserving property. If an NLE can resist the theoretically-grounded attack [25] instantiated under $s_{TG}(\cdot)$, then there is $\mathrm{Pr_D}(v)=\mathrm{Pr_V}(v)$ and $s_2(v)=\mathrm{Pr_V}(v)^{\frac{1}{n}}$ for any v. In this case, it is obvious that $s_2(v_i)\geq s_2(v_j)$ is true if $s_{TG}(v_i)\geq s_{TG}(v_j)$ holds. Therefore, we mainly prove the other side.

Since $\mathrm{Pr_D}(v)$ only depends on the selected vault v, i.e., without impacts from other undecoded vaults, the general solution is of the form $\mathrm{Pr_D}(v_i)=\frac{\mathrm{Pr_V}(v_i)^y}{\sum_{j=1}^{n}\mathrm{Pr_V}(v_j)^y}$, where y is undetermined. Based on Eq. 21, the following relation should hold for any $\mathrm{Pr_V}(v)$:

$$\begin{aligned}&\mathrm{Pr_V}(v_i)^{\frac{n+1}{n}}\left(\sum_{j=1}^{n}\mathrm{Pr_V}(v_j)\right) = \mathrm{Pr_V}(v_i)^y\left(\sum_{j=1}^{n}\mathrm{Pr_V}(v_j)\right)^{\frac{1}{n}},\\ &\text{i.e.,}\quad \sum_{j=1}^{n}\mathrm{Pr_V}(v_i)^{\frac{n+1}{n}-y}=1 \quad\text{or}\quad \sum_{j=1}^{n}\mathrm{Pr_V}(v_i)^y=1.\end{aligned} \quad (23)$$

This means that $y=1$ or $y=1/n$. In both cases, we have $\mathrm{Pr_V}(v_i)\geq\mathrm{Pr_V}(v_j)$ if $s_2(v_i)\geq s_2(v_j)$ holds. Since $s_{TG}(v)=s_2(v)\cdot\mathrm{Pr_V}(v)^{\frac{n-1}{n}}$, we have $s_{TG}(v_i)\geq s_{TG}(v_j)$ if $s_2(v_i)\geq s_2(v_j)$ holds. This ends the proof. □

B.2 Proofs of Criterion #2

Before proving Theorem 3 in Sec. 3.3, we first formally define *single* rules belonging to a password: They are the rules that can be directly retrieved from the UPPM grammars for unique passwords.

Definition 3. *Suppose a password* pw *can be parsed into the rule vector* $\mathbf{rv}=(\mathsf{S}\to\mathcal{T}_{pw}=\prod_{m\in[1,M]}\mathsf{T}^{(m)}_{l_{m,i,j}},\mathsf{T}^{(m)}_{l_{m,i,j}}\to\mathtt{str})$, *where* (m,i,j) *identifies the position of the segment* $\boldsymbol{\omega}=\boldsymbol{\omega}(m,i,j,\mathtt{str},\mathcal{T}_{pw})$ *in the vault and* $\mathsf{T}^{(m)}_{l_{m,i,j}}$ *denotes the template of the length* $l_{m,i,j}$. *No two consecutive* m *values are the same. Particularly,* m *denotes the type of segment,* $i\in[1,|v|]$ *denotes the index of single password that the content of* $\boldsymbol{\omega}$ *belongs to, i.e.,* $\boldsymbol{\omega}(\mathtt{str})\subset pw_i$, *and* $j\in[1,j_{m,i}]$ *denotes the* j*-th type-*m *segment in* pw_i, *with* $j_{m,i}$ *being the number of type-*m *segments in* pw_i. *A single rule of* $\boldsymbol{\omega}$ *is of the form* $\mathbf{r}=(\mathsf{S}\to\mathsf{T}^{(m)}_{l},\mathsf{T}^{(m)}_{l}\to\mathtt{subcont})$, *where* $1\leq l\leq l_{m,i,j}$ *and* subcont *is the subsegment content with* $\mathtt{subcont}\subset\mathtt{str}$.

Algorithm 1: Finding all single rules parsing all types of password segments in a vault (Step #1).

Input : Password vault v and ordered PPM grammars $(\mathbf{r}, O)$, where $\mathbf{r}=(\mathsf{S}\to\mathsf{T}^{(m)}_{l},\mathsf{T}^{(m)}_{l}\to\mathtt{str})$.
Output: Ranked single rule set $\mathbf{R}(\boldsymbol{\omega},l)$ for each segment $\boldsymbol{\omega}$ in v under each type and length parameters m and l.

begin
  for *Unique* $pw\in v$ **do**
    $(\mathsf{S}\to\mathsf{T}^{(m)}_{\mathtt{Len(cont)}},\mathsf{T}^{(m)}_{\mathtt{Len(cont)}}\to\boldsymbol{\omega}(m,i,j,\mathtt{str},\mathcal{T}_{pw}))$ $\leftarrow$ Parse$(pw,\mathsf{T}^{(1)},\ldots,\mathsf{T}^{(m)})$; /* Each $pw\in v$ is parsed into rules of disjoint segment types. The index i means the i-th unique password in v, j means the j-th type-m segment in pw_i, and cont denotes segment contents.*/
  for $m\in[1,M]\wedge\boldsymbol{\omega}(m')==m$ **do**
    for l=1 **to** Len$(\boldsymbol{\omega}(\mathtt{str}))$ **do**
      Subseg(m,l):=**Subseg**$(\boldsymbol{\omega},l)$:=$\emptyset$; /*__Subseg__(m,l) is the subsegments of the type-m and length l, and __Subseg__$(\boldsymbol{\omega},l)$ is the subsegments of $\boldsymbol{\omega}$.*/
      Subseg$(\boldsymbol{\omega},l)$:=$\{\boldsymbol{\mu}(m,i,j,\mathtt{subcont},s,t,\mathcal{T}_{pw})|\ \boldsymbol{\mu}(m)=\boldsymbol{\omega}(m),\boldsymbol{\mu}(i)=\boldsymbol{\omega}(i),\boldsymbol{\mu}(j)=\boldsymbol{\omega}(j),\mathtt{subcont}\subset\mathtt{str}\}\leftarrow$ Findsubseg$(\boldsymbol{\omega},m,l)$; /*Finding all subsegments of the length l, where s (resp. t) is the number of character(s) in cont before (resp. after) the first (resp. last) character in subcont, and $s+l+t$=Len$(\boldsymbol{\omega}(\mathtt{str}))$. */
      Subseg(m,l):=**Subseg**$(m,l)\bigcup$**Subseg**$(\boldsymbol{\omega},l)$;
      Rank all $\boldsymbol{\mu}\in$**Subseg**(m,l) in order $(O,(i,j))$;
      for $\boldsymbol{\mu}\in$**Subseg**(m,l) **do**
        for $(\mathsf{S}\to\mathsf{T}^{(m)}_{l},\mathsf{T}^{(m)}_{l}\to\boldsymbol{r})\in$**PPM**$(m,l)$ **do**
          if $\boldsymbol{\mu}(\mathtt{subcont})==\mathbf{rv}(\mathtt{str})$ **then**
            $\mathrm{Pr_V}(\mathsf{S}\to\mathsf{T}^{(m)}_{l},\mathsf{T}^{(m)}_{l}\to\boldsymbol{\mu})\leftarrow$ $\mathrm{Pr_D}(\mathsf{S}\to\mathsf{T}^{(m)}_{l},\mathsf{T}^{(m)}_{l}\to\mathbf{rv})$; /*Checking if subcont can be parsed by a single rule.*/
          $\boldsymbol{\mu}$:=$(\boldsymbol{\mu},\mathrm{Pr_V}(\mathsf{S}\to\mathsf{T}^{(m)}_{l},\mathsf{T}^{(m)}_{l}\to\boldsymbol{\mu}))$;
      R$(\boldsymbol{\omega},l)$:=Group$(\mathsf{S}\to\mathsf{T}^{(m)}_{l},\mathsf{T}^{(m)}_{l}\to\boldsymbol{\mu}(i,j))$;

We now derive Alg. 1 to fulfill Step #1 in Sec. 3.3. At a high level, Alg. 1 first partitions passwords in the vault into disjoint types of segments, *with each segment as long as possible,* and identifies the rule vectors generating them via the UPPM grammars. Then, for all segments of each given type, Alg. 1 finds all its subsegments under each length l, and these segments are ranked in the (O,i,j)) order. After this, Alg. 1 checks if subsegments can be parsed by single rules given by the UPPM, and records their probabilities if so.

Finally, single rules of the same segments are grouped together. The order (O, i, j) in Line 8 gives the descending priority for segment content, the index i and the index j in ranking single rules, i.e.,

$$\begin{aligned}&\textsc{Rank}_{(O,i_1,j_1)}(\mathtt{str}, i_1, j_1) < \textsc{Rank}_{(O,i,j)}(\mathtt{str}_2, i_2, j_2)\\ \Longleftrightarrow &\begin{cases}\text{If} & \textsc{Rank}_O(\mathtt{str}_1) < \textsc{Rank}_O(\mathtt{str}_2)\\ \text{If} & \mathtt{str}_1 = \mathtt{str}_2 \wedge i_1 < i_2\\ \text{If} & \mathtt{str}_1 = \mathtt{str}_2 \wedge i_1 = i_2 \wedge j_1 < j_2.\end{cases}\end{aligned} \tag{24}$$

We now can derive Lemma 2 for the time complexity of Step #1.

Lemma 2. *Alg. 1 can find all single rules, with those parsing the same password segment ranked in the same PPM order. The time complexity is* $O\Big(\sum_{m,i,j}\sum_{l=1}^{l_{m,i,j}} n_{m,l}+n'_{m,l}\cdot l\cdot(l_{m,i,j}-l)\Big)$, *where* $n'_{m,l}$ *(resp.* $n_{m,l}$*) is the number of (sub)segments of type-m and length* l *in the given vault* v *(resp. given by the UPPM grammars).*

Proof. The computation-intensive operations are parsing passwords in Line 3, finding all subsegments in Line 7, ranking subsegments in Line 9, and checking if they can be generated by the UPPM grammars in Lines 10~14. *Here, we suppose all single rules have been ranked in the order* O *in the precomputation.*

More specifically, the time complexity of parsing the vault is $O(\sum_{m,i,j} l_{m,i,j})$. For each $\boldsymbol{\omega}=(m,i,j,\mathtt{str},\mathcal{T}_{pw})$ segment of length $l_{m,i,j}=\text{Len}(\boldsymbol{\omega}(\mathtt{str}))$, it has $l_{m,i,j}-l+1$ subsegments of length l, and the time complexity of ranking them in the order (O,i,j) using the radix sort approach is $O(n'_{m,l}(l_{m,i,j}-l+1)(l+2))$. In this case, we obtain ranked subsegments $\{\boldsymbol{\mu}_k\}_{k=1}^{n'_{m,l}(l_{m,i,j}-l+1)}$ (duplicated) of length l, which are substrings of the length-l segments parsed by the UPPM. Since both contents of $\{\boldsymbol{\mu}_k\}$ and UPPM-parsing segments $\mathbf{rv}(\mathtt{str})$ are orderly ranked, the time complexity of checking if a subsegment can be generated by a single rule (i.e., existing in the UPPM grammars) is $O(n_{m,l})$. Finally, the time complexity of grouping single rules $\mathbf{r}_{\boldsymbol{\mu}}=(\mathtt{S}\rightarrow\mathtt{T}_l^{(m)},\mathtt{T}_l^{(m)}\rightarrow\boldsymbol{\mu})$ based on the (i,j) index is $O(n'_{m,l}(l_{m,i,j}-l+1))$. From Eq. 24, $\mathbf{r}_{\boldsymbol{\mu}}\in\mathbf{R}(\boldsymbol{\omega},l)$ are ranked in the order O. Thus, the time complexities are $O(\sum_{m,i,j} l_{m,i,j})$ for parsing rules, $O(\sum_{m,i,j}\sum_{l=1}^{l_{m,i,j}} n'_{m,l}(l_{m,i,j}-l+1)(l+2))$ for finding all subsegments, $O(\sum_{m,i,j}\sum_{l=1}^{l_{m,i,j}} n_{m,l})$ for checking, and $O(\sum_{m,i,j}\sum_{l=1}^{l_{m,i,j}} n'_{m,l}(l_{m,i,j}-l+1))$ for grouping, respectively. Therefore, the overall time complexity is $O(\sum_{m,i,j}\sum_{l=1}^{1,l_{m,i,j}} n_{m,l}+n'_{m,l}\cdot l\cdot(l_{m,i,j}-l))$. □

In Step #2 (see Sec. 3.3), our improved generic IS-PMTE needs to find all single rule combinations parsing each segment $\boldsymbol{\omega}$ and their probabilities. *Note that if single rules can parse* $\boldsymbol{\omega}$ *of length* $l_{m,i,j}$ *into* k *disjoint subsegments, there must exist* $d\in[1,l_{m,i,j}]$ *that can parse its length* $l_{m,i,j}-d$ *subsegments into shorter* $k-1$ *subsegments.* Therefore, Step #2 can be recursively achieved by dynamic programming, and we show the process in Alg. 2. To find its time complexity, we introduce the generating function and derive Lemma 3.

Definition 4. *The generating function of is* $A(x)=\sum_{n=0}^{\infty} a_n x^n$, *where* a_n *is the number of objects of size n in the class* $\mathcal{A}$.

Lemma 3. *For a segment* $\boldsymbol{\omega}(m,i,j,\mathtt{str},\mathcal{T}_{pw})$ *and length* $l_{m,i,j}$ *(i.e.,* $\text{Len}(\boldsymbol{\omega}(\mathtt{str}))=l_{m,i,j}$*), the time complexity of finding all rule vectors parsing all segments along with their probabilities is* $O\big(\sum_{m,i,j} 2^{l_{m,i,j}}\big)$, *and there exist* $O\big(2^{l_{m,i,j}}\big)$ *number of rules in each* $\mathbf{RV}(\boldsymbol{\omega})$.

Proof. We denote T_L as the time complexity of finding all rules of the subsegment of $\boldsymbol{\omega}(\mathtt{str})[1,L]$, so $T_{l_{m,i,j}}$ is the time complexity of finding all rules in $\mathbf{RV}(\boldsymbol{\omega})$ for $\boldsymbol{\omega}$. Suppose the first $L-l$ characters have been well parsed by single rule combinations, and we append them to parse the rest l characters. Since all single rules obtained from Alg. 1 are listed orderly, the search, comparison and appending operations in Lines 6, 7, 10 and 11 in Alg. 2 only take $o(1)$ time complexity for each given l. Therefore, we have $T_L=\sum_{l=1}^{L-1}(T_{L-l}+c)$, where c is a constant. To find the general form of T_L, we construct the generating function as $T(x)=\sum_{L=1}^{\infty} T_L\cdot x^L$, and we have

$$\begin{aligned}T(x)&=\sum_{L=1}^{\infty}\Big(\sum_{l=1}^{L-1}T_{L-l}\Big)x^L+c\sum_{L=1}^{\infty}(L-1)x^L\\&=\sum_{L=1}^{\infty}\Big(\sum_{l=1}^{L-1}T_l\Big)x^L+c\sum_{L=1}^{\infty}L\cdot x^L-c\sum_{L=1}^{\infty}x^L\\&=\sum_{L=1}^{\infty}\Big(\sum_{l=L}^{\infty}x^{l+1}\Big)T_L+c\cdot x\frac{\mathrm{d}}{\mathrm{d}x}\Big(\sum_{L=0}^{\infty}x^L\Big)-c\sum_{L=1}^{\infty}x^L\\&=\frac{x}{1-x}T(x)+\frac{cx^2}{(1-x)^2}.\end{aligned} \tag{25}$$

Thus, $\frac{1-2x}{1-x}T(x)=\frac{cx^2}{(1-x)^2}$, and $T(x)$ can be solved as $T(x)=c\cdot\sum_{L=1}^{\infty}(2^{L-1}-1)x^L$. This means that $T_L=2^{L-1}-1$, so the time complexity is $O(\sum_{m,i,j}2^{l_{m,i,j}})$ for finding all single rule combinations for segments $\boldsymbol{\omega}(m,i,j,\mathtt{str},\mathcal{T}_{pw})$. Besides, $|\mathbf{RV}(\boldsymbol{\omega})|$ follows the same recursive manner, with $0<c\leq 1$, so there exists $O\big(2^{l_{m,i,j}}\big)$ number of rules. □

Algorithm 2: Finding all parsing rules of all segments by combining single rules (Step #2).

Input : Ranked single rule set $\mathbf{R}(\boldsymbol{\omega},l)$ obtained in Alg. 1.
Output: All generation rules $\mathbf{RV}(\boldsymbol{\omega})$ parsing each $\boldsymbol{\omega}$.

```
1  begin
2      for m ∈ [1, M] ∧ ω(m′)==m do
3          for ω(m′)==m∧L = 1 to Len(ω(str)) do
4              RV(ω):=RV(ω, L):=∅; /* Initialization.*/
5              for l← 1 to L − 1 do
6                  for μ ∈ R(ω, l) do
7                      if μ(subcont) == ω(str)[L − l + 1, L] then
8                          /*ω(str)[L−l + 1, L] means the
                           subsegment content from the L − l + 1-th
                           and ending at the L-th characters.*/
9                      for rv∈RV(ω, L − l) do
10                         RV(ω, L).Append(r, (S→T_l^(m), T_l^(m) →
                           μ(subcont))); /*Appending single rules
                           generating a length l subsegment.*/
11                         ω(Pr_V(r)):=ω(Pr_V(r))·Pr_V(S →
                           T_l^(m), T_l^(m) → μ(subcont));
12             RV(ω):=RV(ω, Len(ω(str))); /*Finding generation of
                the entire ω segment. */
```

After finding all rule vectors $\mathbf{RV}(\boldsymbol{\omega})$ of $\boldsymbol{\omega}$, our improved IS-PMTE aims to fulfill Step #3, i.e., finding all rule vectors $\mathbf{rv}_v\in\mathbf{RV}(v)$ parsing the vault v, along with the probability of each $\mathbf{rv}_v$ in $\mathbf{RV}(v)$. To do this, the improved IS-PMTE concatenates $\mathbf{rv}_{\boldsymbol{\omega}}$ in all $\mathbf{RV}(\boldsymbol{\omega})$ to generate each rule vector $\mathbf{rv}_v$, i.e., $\mathbf{rv}_v=(\mathbf{r})_{\mathbf{r}\in\mathbf{R}(\boldsymbol{\omega}),\boldsymbol{\omega}\in v}$. Accordingly, the

Algorithm 3: Finding all rule vectors of the vault (Steps #3).

Input : Rule vector set $\mathbf{RV}(\boldsymbol{\omega})$ of each segment $\boldsymbol{\omega}$ in Alg. 2.
Output: Rule vector set $\mathbf{RV}(v)$ of the vault v.

begin
  $\mathbf{RV}(v) \leftarrow \emptyset$; /* Initialization.*/
  for $i = 1$ **to** $|v|$ **do**
    $\mathbf{RV}(pw_i) \coloneqq \emptyset$;
    $(\mathsf{S} \rightarrow \mathsf{T}^{(m)}_{i_{l_m}}, \mathtt{T}_{i}{}^{(m)}_{i_{l_m}} \rightarrow \boldsymbol{\omega}(m, i, j, \mathtt{str}, \mathcal{T}_{pw}))$
      $\leftarrow \textsc{Parse}(pw, \mathsf{T}^{(1)}, \ldots, \mathsf{T}^{(M)} = \prod_{m\in[1,M]} \mathsf{T}^{(m)}_{l_{m,i,j}})$;
      /*Finding password parsing way given in Alg. 1 for each password in the vault. */
    for $\mathsf{T}^{(m)}_{i_{l_m}} \in \mathcal{T}_{pw_i}$ **do**
      for $j = 1$ **to** $j_{m,i}$ **do**
        for $\mathbf{rv} \in \mathbf{RV}(\boldsymbol{\omega})$ **do**
          $\mathrm{CDF}(m, i, j) \coloneqq 0$;
          if $\boldsymbol{\omega}(m') == m \wedge \boldsymbol{\omega}(i') == i \wedge \boldsymbol{\omega}(j') == j$ **then**
            $\mathbf{RV}(v)[i, m, j].\textsc{Append}(\mathbf{r})$;
            $\mathrm{CDF}(i, m, j) \coloneqq \mathrm{CDF}(i, m, j) + \mathrm{Pr}_{\mathrm{V}}(\mathbf{r})$;
    $\mathbf{rv}_{pw_i} \coloneqq \prod_{m,j} \mathbf{r}$, $\mathrm{Pr}_{\mathrm{V}}(\mathbf{r}_{pw_i}) \coloneqq \prod_{m,j} \mathrm{Pr}_{\mathrm{V}}(\mathbf{r})$;
    $RNer_i \coloneqq \prod_{m,j} \mathrm{CDF}(i, m, j)$; /*Computing the normalizer.*/
    $\mathbf{RV}(pw_i) \coloneqq \{(\mathbf{r}_{pw_i}, \mathrm{Pr}_{\mathrm{V}}(\mathbf{r}_{pw_i})/RNer_i)\}_{m,j}$;
  $\mathbf{rv}_v \coloneqq \prod_{i=1}^{|v|} \mathbf{r}_{pw_i}$, $\mathrm{Pr}_{\mathrm{V}}(\mathbf{rv}_v) \coloneqq \prod_{i=1}^{|v|} \mathrm{Pr}_{\mathrm{V}}(\mathbf{r}_{pw_i})$; /*Concatenating rule vectors for the entire vault. */
  $RNer \coloneqq \prod_i RNer_i$; /*Calculating the normalizer for v. */
  $\mathbf{RV}(v) \coloneqq \{(\mathbf{rv}_v, \mathrm{Pr}_{\mathrm{V}}(\mathbf{rv}_v)/RNer)\}$;

probability of $\mathbf{rv}_v$ is obtained by multiplying all chosen $\mathbf{rv} \in \mathbf{RV}(\boldsymbol{\omega})$ together (with duplications), i.e., $\mathrm{Pr}_{\mathrm{V}}(\mathbf{rv}_v) = \prod_{\boldsymbol{\omega}\in v, \mathbf{r}\in\mathbf{R}(\boldsymbol{\omega})} \mathrm{Pr}_{\mathrm{V}}(\mathbf{r})$. Further, to find the probability of each $\mathbf{rv}_v \in \mathbf{RV}(v)$, we need to compute the renormalizer, i.e., $\sum_{\mathbf{rv}_v} \mathrm{Pr}_{\mathrm{V}}(\mathbf{rv}_v)$. Since each $\boldsymbol{\omega}$ is independently generated by $\mathbf{rv} \in \mathbf{RV}(\omega)$, we have

$$\sum_{\mathbf{rv}_v} \mathrm{Pr}_{\mathrm{V}}(\mathbf{rv}_v) = \prod_{\boldsymbol{\omega}\in v} \sum_{\mathbf{r}\in\mathbf{R}(\boldsymbol{\omega})} \mathrm{Pr}_{\mathrm{V}}(\mathbf{r}) \tag{26}$$

Thus, we can sum all probabilities of all rules parsing the segment $\boldsymbol{\omega}$, and multiply summed probabilities across segments in the vault to obtain the renormalizer. The complexity is $O(\sum_{\boldsymbol{\omega}\in v} |\mathbf{R}(\boldsymbol{\omega})|)$, i.e., $O(\sum_{m,i,j} 2^{l_{m,i,j}})$. Formally, we have Alg. 3 to fulfill Step #3. In total, by summing up the results in Lemmas 2 and 3 and above, we have Theorem 3 in Sec. 3.3 from the overall time complexity.

Theorem 3. *Under the premises of segment-based UPPMs in Sec. 2.2, the time complexity of finding all rule vectors parsing the vault via Steps #1~#3 is* $O\Big(\sum_{m,i,j} (\sum_{l=1}^{l_{m,i,j}} n_{m,l} + 2^{l_{m,i,j}})\Big)$, *where* $n_{m,l}$ *is the number of the length* l *and type-*m *segments in the UPPM grammars.*

C Additional Results regarding Criteria #2&#3

We present additional information of reused password probability models (RPPMs) and secure incremental update mechanism. In Table 4, we list password similarity features considered in Sec. 3.2.

We first introduce Cheng et al.'s RPPM [19]. In general, their RPPM aims at modifying the head and tail parts of a password, and adopts three types of modifications: delete, insert, and delete-then-insert modifications without allowing their combinations [19]. Particularly, when modifying a password, their RPPM first determines the head or tail part, then the type of modification, and finally the number and content of each atomic modification (i.e., deleting/inserting one character). Their model uses HDN, HIN, TDN, TIN to denote head-delete, head-insert, tail-delete, and tail-insert modification numbers, respectively. For insertion contents, HIC and TIC denote the character of head and tail insertions, respectively. To satisfy the LCSStr constraint between unique and reused passwords, head deletions will be executed first, followed by tail deletions, head insertions, and tail insertions. In this way, the following relations hold for the number of each type of modification:

$$\begin{cases} \mathrm{HDN} \le l_{uw}/2,\ \mathrm{TDN} \le l_{uw}/2 - \mathrm{HDN}, \\ \mathrm{HIN} \le l_{uw}/2 - \mathrm{HDN} - \mathrm{TDN},\ \mathrm{TIN} \le l_{uw} - \mathrm{HDN} - \mathrm{TDN} - \mathrm{HIN}, \end{cases} \tag{27}$$

where l_{uw} is the length of the modified unique password. We use a concrete example to show the process. When modifying "`apple!`" to "`appLE*`", their RPPM first identifies the tail part. Then, it determines the delete-and-insert modification with the number of atomic modifications being 6. Finally, it executes deleting "`!`" and "`e`" and "`1`", and inserting "`L`", "`E`" and "`*`". Thus, the probability is:

$$\begin{aligned} &\mathrm{Pr}_{\mathrm{PW}}(\mathtt{appLE*|apple!}) = \mathrm{Pr}_{\mathrm{RPPM}}(\mathrm{Tail}) \cdot \mathrm{Pr}_{\mathrm{RPPM}}(\text{Delete-then-insert}) \cdot \\ &\mathrm{Pr}_{\mathrm{RPPM}}(\mathrm{TDN}=3) \cdot \mathrm{Pr}_{\mathrm{RPPM}}(\mathrm{TIN}=3) \cdot \mathrm{Pr}_{\mathrm{RPPM}}(\mathrm{TIC}=\mathtt{L}) \cdot \\ &\mathrm{Pr}_{\mathrm{RPPM}}(\mathrm{TIC}=\mathtt{E})\mathrm{Pr}_{\mathrm{RPPM}}(\mathrm{TIC}=\mathtt{*}), \end{aligned} \tag{28}$$

where $\mathrm{Pr}_{\mathrm{RPPM}}(\cdot)$ comes from the trained model.

We now show why their RPPM is vulnerable to encoding attacks, and how we address this issue. At a high level, the modification types are determined before the atomic modification numbers and contents, so any of HDN, HIN, TDN, and TIN cannot be zero for a real vault. However, in decoding, these numbers constrained by Eq. 27 can be decoded as 0, which signals a decoy vault. For a better illustration, we use the following concrete example.

For the password pair (`1234pass`, `pass1235`) in the vault that satisfies the LCSStr feature, if it is real, in encoding, modification types are determined in advance before the numbers and detailed contents. In this case, the types head-delete and tail-insert will be encoded first, and followed by the numbers HDN=TIN=4, and the tail-insert content "`1`", "`2`", "`3`" and "`5`". However, in decoding, the modification types will also be decoded first, so it is possible to make the code decoded as head-delete-then-insert, with HDN=4 and HIN=0. Similar things can also happen in the tail part, where delete-then-insert coexists with TDN=0 and TIN=4. This leads to inconsistency so attackers can identify such vaults as decoys.

To address this issue, in our RPPM, we only encode modification numbers and the corresponding contents, i.e., HDN, HIN, HIC, TDN, TIN and TIC. *We do not encode any specific modification types, since they can be identified by the specific modification results after the decoding process is completed.* We also add the relationship that HDN, HIN, TDN, and TIN cannot be 0 simultaneously. In this way,

Table 4: Similarity features in [19] for determining reused passwords in the real vault and decoy vaults.

Feature	Description
LCSStr	The longest common substring between the two passwords is at least half of the maximum length.
Levenshtein	The Levenshtein (edit) distance between the two passwords is at most half of the maximum length.
LCS	The length of the longest common subsequence (not necessarily consecutive) of the two passwords is at least half of the maximum length of them.
Manhattan	The Manhattan distance between the two passwords is at most half the sum of their lengths.
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.

Table 5: Basic information of mainstream password managers/vaults.

Name	Version	Size	Leak detect.†	Websites	Name	Version	Size	Leak detect.†	Websites
Bitwarden	2025.3.2	16.35MB	✓	https://tinyurl.com/yc4bw7fc	Avira PM	2.21.0.5015	8.48MB		https://tinyurl.com/yvkf2f7b
1Password	8.10.70.27	21.86MB	✓	https://tinyurl.com/4m6f4nc4	McAfee True Key	5.0.0.227	14.38MB	✓	https://tinyurl.com/3mfn8wd9
1Password Beta	8.10.72.25	21.24MB	✓	https://tinyurl.com/s7c4uced	Padloc	4.3.0	5.23MB		https://tinyurl.com/m97td46y
1Password Nightly	8.10.74.12	21.47MB	✓	https://tinyurl.com/47sxtuw4	Password Boss	5.5.5138	461.4KB		https://tinyurl.com/3xpc5td9
LastPass	4.140.2	29.30MB	✓	https://tinyurl.com/2ze2a7xj	Multipassword	0.98.39	2.73MB		https://tinyurl.com/3485t7vw
Dashlane	6.2514.0	14.69MB	✓	https://tinyurl.com/y4yh7nf7	Passwarden	1.3.6	10.59MB		https://tinyurl.com/3pxknzz9
RoboForm	9.6.14.0	6.97MB	✓	https://tinyurl.com/3zrdj6k7	Delinea Web PM	3.11.4	1.19MB		https://tinyurl.com/yc8mecvn
Keeper	17.0.1	87.72MB	✓	https://tinyurl.com/2yxsddhb	LogMeOnce	7.10.2	5.41MB	✓	https://tinyurl.com/5n98jz68
NordPass	5.29.7	15.76MB	✓	https://tinyurl.com/3328c7hn	Passbolt	4.12.0	5.18MB	✓	https://tinyurl.com/y6r35wy6
Zoho Vault	5.1.1	2.89MB		https://tinyurl.com/3heht2by	Iron Vest	10.1.24	6.05MB		https://tinyurl.com/a5dcvrax
KeePass	0.8.5	275.7KB	✓	https://tinyurl.com/47t3ahkw	SafeInCloud PM	24.14.2	2.04MB	✓	https://tinyurl.com/yc2c7686
KeePassXC	1.9.7	993KB	✓	https://tinyurl.com/2ajzydms	Password Safe	1.1.8	1.02MB		https://tinyurl.com/2y52pev3
Dropbox PM	3.49.0	4.93MB		https://tinyurl.com/397dem4w	SPHINX	1.13	21.27KB		https://tinyurl.com/r4smfek6
DualSafe PM	1.4.35	3.10MB		https://tinyurl.com/yk3wtthj	iCloud Keychain	3.0.10	296.8KB	✓	https://tinyurl.com/mpbujewv
Enpass	6.11.3	22.34MB	✓	https://tinyurl.com/5ax4ca6a	TeamPass	4.1.9	1.58MB		https://tinyurl.com/2fyv8dxj
Norton PM	8.2.3.809	15.25MB		https://tinyurl.com/mp94hdw8	Keeweb	4.0.6	2.65MB		https://tinyurl.com/4kec27ap
Gopass	2.1.0	136.31KB		https://tinyurl.com/mw6btjju	Lesspass	11.0.10	621.8KB		https://tinyurl.com/2s35hm9s

† Leakage detection denotes whether a password manager/vault can check whether a user's password has been leaked through services like HIBP [6].

no information about modification types will be decoded, and thus our RPPM can avoid ambiguities in their scheme.

Secure IU mechanism. We then show how to securely update a newly registered password. In 2021, Cheng et al. [19] proposed the incremental update (IU) mechanism that each time updates only one (e.g., the newly added) password. For the old and new versions of the vault v^o and v^n, their encoded codes s_v^o and s_v^n have the same prefix to resist intersection attacks. However, in 2024, Duan et al. [25] exploited the incompleteness in decoy vaults and designed a feature attack with a success rate of nearly 90%. The basic idea is that passwords decoded from the prefix of the real vault should be complete. For example, if a vault decoded from the string prefix contains `passwor` with no end symbol, then its vault is a decoy.

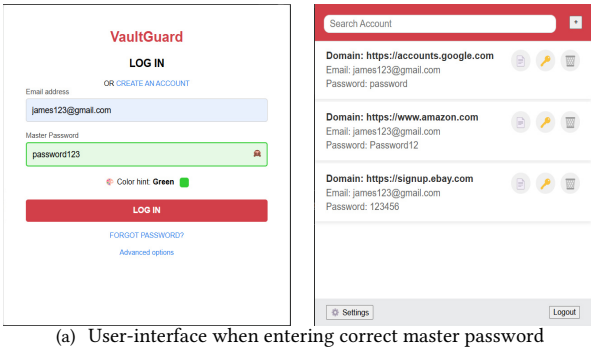
To resist such attacks, all passwords, regardless of their compositions and lengths, should be parsed into rule vectors of the same length (i.e., of the same number of rules). Suppose the maximum password length is l_M. For NLEs with d-gram sequence-based UPPMs, the number of rules is no more than l_M−d+2. We regulate that all l_M−d+2 rules will be used in encoding for each unique password, with empty rules padded with the end symbol.

D Additional Information on Evaluations

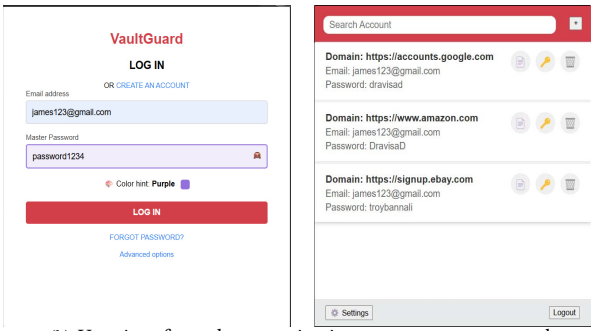
We list off-the-shelf password managers/vaults in Table 5. The sizes of these password managers range from 136KB to 87.72MB, and half of them support password leakage detection services (e.g., HIBP [6]). Since our VaultGuard (with HE-adaptive) takes 15.7MB∼16.6MB (see Table 3), it is highly practical. Password leakage detection services can also be mounted on VaultGuard.

We now demonstrate how we instantiate the color hints that have been suggested in [12, 17, 19] in our VaultGuard solution. We set the color set as {green, blue, yellow, purple, orange}. As shown in Fig. 7, when a user registers with the master password, e.g., "`password124`", a color (e.g., green) depending on the salted hash of the master password will appear as the user's hint. After entering the correct master password, the user can enter the vault and see all passwords stored in it. When the attacker $\mathcal{A}$ attempts to log in with "`password123`", another color hint will occur (e.g., purple), and a decoy vault will occur. This confuses $\mathcal{A}$ and forces her to do online verifications to identify whether it is real.

We also show the average ranks of VaultGuard (with HE-Adaptive) under $n\in\{100, 500, 5000, 10^4\}$ in Table 6. Compared with n=1,000, the maximum difference is as small as 0.35%∼3.20%. This substantiates the claim that the evaluation results obtained under



(a) User-interface when entering correct master password



(b) User-interface when entering incorrect master password

Figure 7: The user-interface of our VaultGuard solution (see Sec. 5.2) integrated color hints as suggested in [12, 17, 19].

Table 6: Average ranks of VaultGuard under different n.†

Honey vault distinguishing attack	Average rank $\bar{r}$ value				max $\bar{r}$ diff.‡
	n=100	n=500	n=5,000	n=10^4	
KL divergence attack [29]	51.30%	51.21%	51.01%	50.98%	0.77%
Single password attack [19]	47.50%	47.72%	47.33%	47.26%	0.67%
Password similarity attack [19]	48.48%	49.62%	49.40%	49.33%	0.74%
Theoretically-grounded attack [25]	40.41%	40.63%	41.15%	40.80%	0.64%
Weak encoding attack [20]	53.66%	52.96%	50.61%	53.70%	3.20%
Strong encoding attack [20]	49.28%	48.33%	48.47%	49.22%	0.97%
Adaptive extra attack [19]	49.63%	49.07%	48.83%	49.53%	1.25%
(Adaptive) hybrid attack [19]	49.84%	49.68%	49.49%	50.14%	0.35%

† VaultGuard means VaultGuard-NLE with HE-Adaptive (see Sec. 5.1).
‡ The maximum difference in the last column is calculated as the maximum difference of average ranks taken under $n\in\{100, 500, 5000, 10^4\}$ and n=1, 000 (see Table 2) against the same honey vault distinguishing attack.

n=1,000 are representative [17, 29], and can be applied to other n depending on users' master password strengths.