Probability Model Transforming Encoders Against Encoding Attacks

Haibo Cheng, Zhixiong Zheng, Wenting Li, and Ping Wang, *Peking University;*
Chao-Hsien Chu, *Pennsylvania State University*

https://www.usenix.org/conference/usenixsecurity19/presentation/cheng

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

Probability Model Transforming Encoders Against Encoding Attacks

Haibo Cheng[†,‡], Zhixiong Zheng[†,‡], Wenting Li[†,‡], Ping Wang[†,‡,⋆], Chao-Hsien Chu[§]
[†]*Peking University,* {hbcheng,zxzheng,wentingli,pwang}@pku.edu.cn
[‡] *Key Laboratory of High Confidence Software Technologies (PKU), Ministry of Education, China*
[§]*Pennsylvania State University,* chu@ist.psu.edu
[⋆]*Corresponding author*

Abstract

Honey encryption (HE) is a novel encryption scheme for resisting brute-force attacks even using low-entropy keys (e.g., passwords). HE introduces a *distribution transforming encoder (DTE)* to yield plausible-looking decoy messages for incorrect keys. Several HE applications were proposed for specific messages with specially designed *probability model transforming encoders (PMTEs)*, DTEs transformed from probability models which are used to characterize the intricate message distributions.

We propose attacks against three typical PMTE schemes. Using a simple machine learning algorithm, we propose a distribution difference attack against genomic data PMTEs, achieving 76.54%–100.00% accuracy in distinguishing real data from decoy one. We then propose a new type of attack—*encoding attacks*—against two password vault PMTEs, achieving 98.56%–99.52% accuracy. Different from distribution difference attacks, encoding attacks do not require any knowledge (statistics) about the real message distribution.

We also introduce a generic conceptual probability model—*generative probability model (GPM)*—to formalize probability models and design a generic method for transforming an arbitrary GPM to a PMTE. We prove that our PMTEs are information-theoretically indistinguishable from the corresponding GPMs. Accordingly, they can resist encoding attacks. For our PMTEs transformed from existing password vault models, encoding attacks cannot achieve more than 52.56% accuracy, which is slightly better than the randomly guessing attack (50% accuracy).

1 Introduction

Password-based encryption (PBE) is a fundamental scheme in many real-world systems for file encryption or authentication. However, due to the limitations of human memory, users often use weak passwords [26, 43] and reuse them [11, 30]. This leads to the vulnerability of traditional PBEs (e.g., PKCS #5 [22]) against brute-force attacks (so-called password guessing attacks), including trawling guessing attacks [25, 42] and targeted guessing attacks [29, 41].

Several methods were proposed to address this threat. We summarize these countermeasures into three types. The first type is to increase the complexity of decryption for attackers, including: 1) salting, which pressurizes attackers into enumerating passwords for every user (salt); 2) using special password-hashing functions (e.g., iterated hash functions [22, 33] and memory-hard functions [6, 31]) as the key derivation function (KDF) in PBE, which increases attackers' cost of computing and memory by a constant factor but also consumes legitimate users' extra cost by the same factor. For example, LastPass, a password vault software, utilizes these methods, including salting, 5,000 rounds of PBKDF2-SHA256 on clients and 100,000 rounds on servers [38].

The second type of countermeasures is to harden passwords with other factors (e.g., servers [12, 23], devices [19, 35, 37], biometrics [9, 28]) to generate high-entropy keys. These methods are widely used in authentication protocols, for example, two-factor authentication [20, 36]. Note that LastPass also supports YubiKey devices to secure password vaults [2]. However, these methods need additional devices (servers, biometric readers) and do worse than single password methods on deployability [8]. Besides, if the additional factor gets stolen or lost (without a backup), the message encrypted cannot be recovered (e.g., [23]).

The last type of countermeasures is to generate plausible-looking decoy messages for wrong keys to confuse attackers. Several specific encryption schemes for specific data used this method [5, 17], and Juels and Ristenpart proposed a generic method called Honey Encryption (HE) [21]. HE introduces a distribution-transforming encoder (DTE) and encodes a message following a known distribution to a uniform seed before encrypting. Therefore, plausible-looking decoy messages are generated by DTE decoding incorrect seeds when decrypting a ciphertext under wrong keys. If the DTE is perfectly secure, i.e., decoy messages are indistinguishable from real ones, then attackers enumerating all passwords only get many messages and cannot distinguish the right one. This countermeasure

achieves information-theoretic security without declining on deployability and bringing legitimate users' extra cost.

Owing to the security of HE, several applications of HE [10,14,18] were proposed. In this paper, we focus on three typical ones, including two password vault schemes [10, 14] and one genomic data protection scheme [18]. A password vault contains an individual user's multiple passwords on websites or services and is usually encrypted under a user-chosen password, called master password. Passwords stored in password vaults are of great value (e.g., PINs of credit cards, passwords of virtual currency accounts) and hence greatly attract attackers' attention. Similar to password vaults, genomic data is sensitive and needs long-term protection, as it is unchangeable during one's lifetime and correlated with his relatives.

The key of a HE scheme is to design a secure DTE. It is easy for messages following a simple distribution, for example, a uniform distribution, a normal distribution. Juels and Ristenpart [21] designed a generic purpose DTE *IS-DTE* and several specific DTEs for RSA secret keys. Notwithstanding, it is still a great challenge to design a secure DTE for messages following intricate distributions, e.g., natural language texts, passwords, password vaults, and genomic data. Probability models are usually needed to characterize the message distributions. We call these DTEs *probability model transforming encoders (PMTEs)*, which are transformed from probability models instead of distributions. Note that Chatterjee et al. [10] named DTEs for natural language texts as *natural language encoders (NLEs)*, which are a subset of PMTEs. Though all the existing PMTE schemes [10, 14, 18] are designed for specific messages, it is still of necessity to propose a generic PMTE designing method.

In addition, the security evaluations of PMTEs are not comprehensive. The designers of password vault PMTEs [10, 14] tried to use machine learning algorithms and Kullback-Leibler divergence to distinguish real and decoy vaults, without considering the difference between the real and decoy seeds. For the PMTEs in [18], it evaluated the goodness of probability models with chi-square goodness-of-fit tests, but did not study the influence of their goodness on the security of PMTEs. These issues on PMTE study hinder the widespread use of HE.

1.1 Our Contribution

In order to evaluate the security of PMTEs, we propose a framework with encoding attacks and distribution difference attacks. We show that password vault PMTEs [10, 14] suffer from encoding attacks while genomic data PMTEs [18] cannot resist distribution difference attacks. *Encoding attacks*, which are a new type of attack we propose, do not require any knowledge of real message distributions. The strong encoding attack achieves 98.56%–99.52% accuracy (in distinguishing a real vault from a decoy one) against password vault PMTEs. Meanwhile, using a principal component analysis (PCA) and a support vector machine (SVM) with a radial basis function (RBF) kernel, a distribution difference attack achieves 76.54%–100.00% accuracy against genomic data PMTEs.

We also propose a generic PMTE designing method for arbitrary probability models, by introducing a generic conceptual probability model—*generative probability model (GPM)*—to formalize probability models. We prove that our PMTEs are information-theoretically indistinguishable from corresponding GPMs, which means that they can resist encoding attacks. For our proposed PMTEs of existing password vault models, encoding attacks cannot capture more than 52.56% accuracy, compared with the randomly guessing attack (50% accuracy).

2 Background and Related Works

We introduce the basic concepts of HE as well as three typical HE applications with their specific PMTEs.

2.1 Honey Encryption

Honey Encryption (HE) [21], proposed by Juels and Ristenpart, is a novel encryption scheme using low-entropy keys (e.g., passwords) which resists brute-force attack through generating a plausible decoy message for every incorrect key. To produce decoy messages, HE introduces a randomized encoder, called *distribution transforming encoder (DTE)*.

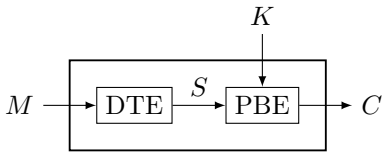


Figure 1: Honey Encryption

Figure 1 shows the encryption progress for a message M. The encryption first encodes M into a seed S by DTE, then encrypts S using PBE with the key K and finally outputs the ciphertext C. The PBE used in HE is a traditional PBE but must satisfy that decrypting any ciphertext under any key yields a valid seed (e.g., AES in CTR-mode with PBKDF). Therefore, decrypting C under an incorrect key K' will yield a wrong seed S' and a decoy message M' by decoding S'. The key of HE is designing a secure DTE which generates indistinguishable decoy messages. Juels and Ristenpart [21] proposed a general purpose DTE *IS-DTE*, for the messages following a simple distribution, such as a uniform distribution.

2.2 Password Vault Schemes

Two HE-based password vault schemes [10, 14] were proposed to resist brute-force attack in the literature. A password vault contains several passwords encrypted under a master

password and hence is a rich target for attackers due to the value of passwords. Using HE, attackers who have stolen an encrypted password vault will get many vaults by enumerating master passwords offline and need to verify the correctness of these vaults online. In contrast to offline guessing, online guessing is more resource-consuming, because it is easily blocked by remote servers with diversities of methods (e.g., login rate limiting [16,32] and malicious login detection [13]). Hence, HE-based password vault schemes have great improvements in security. Moreover, user surveys [11, 24, 30] and empirical experiments on real data [7, 11, 41, 42] showed that users often use weak passwords and reuse passwords on different services and websites. Therefore, designing a PMTE for password vaults needs to characterize the single-password distribution and the similarity between passwords in a vault.

Chatterjee et al. [10] proposed the first HE-based password vault scheme *NoCrack*. They improved a kind of password model—PCFG model [42]—and put forward a sub-grammar approach for the password similarity based on PCFG models. We denote this improved PCFG model as *Chatterjee-PCFG* in this paper. By designing PMTEs for PCFG models and sub-grammars respectively, they presented a PMTE for password vaults. Also, Chatterjee et al. [10] designed PMTEs for another kind of password models—Markov models.

Golla et al. [14] put forward a new PMTE for password vaults. In contrast to Chatterjee et al. [10], Golla et al. used a Markov model [25] for the single-password distribution and a reuse-rate approach for the password similarity. Their PMTE for password vaults combines Chatterjee et al.'s PMTEs for Markov models and IS-DTEs for normal distributions (assuming reuse-rates follow normal distributions). In addition, Golla et al. [14] brought in the concept of adaptive PMTEs (Golla et al. used the word *adaptive NLEs*). By adjusting the Markov model according to the real vault, an adaptive PMTE can generate decoy vaults which are more similar to the real vault. Therefore, these decoy vaults are more difficult to be distinguished from the real one. In this paper, we call adjusted probability models according to the real message *adaptive probability models*, in contrast to *static probability models*.

Chatterjee et al.'s [10] and Golla et al.'s [14] PMTEs for password vaults have the same form named *encode-then-concatenate*. Taking the Chatterjee et al.'s PMTE for a PCFG model as an example, when encoding a password, this PMTE: 1) parses the derivation of the password; 2) encodes each production rule in the derivation to a seed respectively; 3) concatenates these seeds of production rules in order and pads the concatenation to a fix length. Other PMTEs are similar, which encode each character or reuse-rate and then concatenate these seeds.

2.3 Genomic Data Protection Scheme

Genomic data is more sensitive than password vault and needs long-term protection. Once a person's genomic data is compromised, it will affect him during his lifetime and even his relatives, because of the correlation between relatives' genomic data. Huang et al. [18] proposed a genomic data protection scheme called *GenoGuard* based on HE. The genomic data protected by GenoGuard is represented by a sequence of single nucleotide variants (SNVs), which can be viewed as a string of the alphabet $\{0,1,2\}$.

To fit genomic data, Huang et al. [18] evaluated four types of models with chi-square goodness-of-fit tests, including a uniform distribution model, a public LD (linkage disequilibrium) model, three Markov models, and a recombination model. Since the recombination model delivers the best performance, they chose it for GenoGuard. Furthermore, Huang et al. [18] proposed a novel PMTE for these sequences with a different form named *shrink-then-encode*. When encoding, this PMTE shrinks the seed interval for each character in the string according to the probability of the character and randomly picks a seed in the final seed interval.

3 Attacks Against Typical PMTEs

A PMTE is secure (i.e., decoy messages generated by a PMTE are indistinguishable from real ones), if and only if the probability model is accurate for the real message distribution and the PMTE is secure for the probability model. Based on that, we propose a framework to evaluate the security of PMTEs with two types of attacks: 1) *distribution difference attacks* exploiting the difference between the real message distribution and the message probability model (i.e., the decoy message distribution); 2) *encoding attacks* exploiting the difference between the probability model and the PMTE.

3.1 Attacker Model

Attackers that we study in this paper have stolen ciphertext of a message and further want to recover it. Based on the Kerckhoffs's principle, we assume that attackers know the HE algorithm, including DTEs, but do not know the key or any information of the message. It is reasonable because the program shipped to users usually contains the encryption/decryption module. Moreover, this is an essential assumption for an attacker to carry out decrypting. More advanced attackers (e.g., attackers in [10]) may equip themselves with some knowledge about the real message distribution (e.g., the character distribution of messages). However, encoding attackers we employed do not need any information about the real message distribution. Merely relying on the DTEs, such attackers can distinguish real and decoy messages with high accuracy.

To recover the message, attackers: 1) decrypt the ciphertext under N keys $\{k_i\}_{i=1}^{N}$ and get N messages $\{M_i\}_{i=1}^{N}$; 2) choose the most likely message. For some special types of messages which can be verified online, for example, authentication certificates (passwords, password vaults or authentication keys),

Algorithm 1: The attack process to recover a stolen ciphertext.

Input: a stolen ciphertext c, N keys/passwords $\{k_i\}_{i=1}^{N}$ for decryption, and a weight function p.
Output: a guessing list for messages (in decreasing order of p).

```
for i ← 1 to N do
    S_i ← decrypt_{k_i}(c)
    M_i ← decode(S_i)
end
Sort {M_i}_{i=1}^N in decreasing order of p(M_i) (or p(S_i)), then output
the list. /* Different attacks are equipped with
different weight functions p, where p(M_i) usually
reflects the probability that M_i is real.       */
```

attackers need to sort these N messages to minimize the number of online verifications. To characterize attackers in a unified form, we consider an attacker only picking one message also as a sorting attacker who picks the first one in his order. Assuming an attacker sorts the messages in decreasing order of a weight function p, the attack process can be represented as Algorithm 1.

The efficiency of an attacker depends on 1) the guessing order of keys and 2) the sorted order of messages. These two orders correspond to two factors—keys and DTEs, affecting the security of HE schemes. The stronger the keys are, the harder they are to be cracked. Keys used by HE are usually human-memorable passwords. Password researches have attracted great attention recently, such as password guessing [25,41,42], password strength meter [15,39,40], password generation policy [3,34]. However, same as previous literature [10,14], we ignore the influence of keys on the security of HE schemes and only focus on the security of DTEs, i.e., the indistinguishability of decoy messages.

3.2 Analyses of Password Vault PMTEs

Chatterjee et al.'s PMTE [10] for password vaults uses a sub-grammar approach to model the similarity of passwords in one vault. Specifically, the sub-grammar (based on Chatterjee-PCFG) of vault $V = (\text{password}, \text{password1})$ is $\{\texttt{S} \to \texttt{W},\ \texttt{S} \to \texttt{WD},\ \texttt{W} \to \text{password},\ \texttt{D} \to 1\}$, where `W` represents an English word and `D` represents a digit string. In fact, Chatterjee-PCFG is more comprehensive. We simplify it for ease of explanation. To encode a vault, this PMTE 1) first parses the sub-grammar of the vault, 2) then encodes the sub-grammar, and 3) finally encodes the passwords in the vault according to the sub-grammar. Decoding is in the opposite direction.

Because sub-grammars are parsed from the real vaults when encoding, all production rules in sub-grammars are used by passwords in the real vaults. Unfortunately, it may not hold when decoding a random seed. For example, decoding a random seed, the sub-grammar may be $SG = \{\texttt{S} \to \texttt{W},\ \texttt{S} \to \texttt{WD},\ \texttt{W} \to \text{password},\ \texttt{D} \to 1\}$, and the vault may be $V = (\text{password}, \text{password})$. As passwords are generated independently based on sub-grammars when decoding, production rules (e.g., $\texttt{S} \to \texttt{WD}$) in the sub-grammar may not be used by any password in the vault. In addition, decoded sub-grammars may contain identical rules, but encoded ones do not, because the rules are also independently generated when decoding a random seed.

Similar phenomena also appear in Golla et al.'s PMTEs [14]. They used a reuse-rate approach to model password similarity. Given $V = (\text{password1}, \text{password1}, \text{password@})$, Golla et al.'s PMTEs take "password1" as the base password of V and "password@" as a password modified from the base password. When encoding, they 1) encode the base password ("password1") and the reuse-rate of the base password ($\frac{2}{3}$), 2) encode reuse-rates of modified passwords ($\frac{1}{3}$) and the modified characters ("@"). More specifically, Golla et al.'s PMTEs divide the vault into six subsets $\{V_i\}_{i=0}^{5}$: passwords with an edit distance of i to the base passwords V_i ($0 \le i \le 4$) and the remaining passwords V_5. Assuming the proportion (reuse-rate) of V_i in V follow a normal distribution with a small variance, $|V_i|$ (the cardinality of V_i) is encoded by the DTE of the normal distribution, for $0 \le i \le 4$. In addition, the base password, modified characters (of passwords in V_i for $1 \le i \le 4$) and remaining passwords in V_5 are encoded by PMTEs of Markov models.

The sum of $|V_i|$ for $0 \le i \le 4$ (without $|V_5|$) is less than or equal to $|V|$ when encoding. However, it may not hold when decoding a random seed, because proportions of V_i are generated independently. Further, the modified character of password pw in V_i may be the same as the original character of the base password when decoding a random seed, which means pw actually belongs to V_j with $j < i$. But this is not possible when encoding a real vault.

3.3 Attacks Against Password Vault PMTEs

In the above analyses of password vault PMTEs, we dig out some features that real seeds (encoding from real vaults) must have but decoy seeds (random seeds) may not have. Therefore, an attacker is able to exclude some decoy seeds if they do not have these features. Let p_F denote the weight function based on the feature F:

$$p_F(S) = \begin{cases} 1, & \text{if the seed } S \text{ has feature } F, \\ 0, & \text{otherwise.} \end{cases}$$

We now present four features for exploration, the first two features for the Chatterjee et al.'s PMTE [10] and the last two features for Golla et al.'s PMTEs [14]:

1. Feature UR (unused rule): there is no unused rule in the sub-grammar decoded from the seed.
2. Feature DR (duplicate rule): there is no duplicate rule in the sub-grammar decoded from the seed.
3. Feature ED (edit distance): every password in the vault has the same value of i as the one decoded from the seed.

Algorithm 2: The weight function $p_{\text{PCA+SVM}}$ of the PCA+SVM attack

training:
  Input: a dataset *snvsList* containing the same number of real and decoy SNV sequences with the label (0 for decoy and 1 for real) list *labelList*.
  Output: a PCA model *pca* and a SVM model *svm*.
  `/* The classes SVC and PCA we use are svm.SVC and decomposition.PCA in Scikit-learn, a machine learning library for Python. */`
  $pca \leftarrow PCA(n\_components = 10)$ `/* We use the default parameters except n_components as 10. */`
  $pca.fit(snvsList)$
  $reducedSNVsList \leftarrow pca.transform(snvsList)$
  $svm \leftarrow SVC(probability = True)$
  $svm.fit(reducedSNVsList, labelList)$
end
function $p_{\text{PCA+SVM}}(s)$
  Input: an SNV sequence s.
  Output: the SVM-estimated probability that s is real.
  $reducedSNVs \leftarrow pca.transform([s])[0]$
  $p \leftarrow svm.predict\_proba([reducedSNVs])[0,1]$
  return p
end

4. Feature PN (password number): the sum of $|V_i|$ $(0 \le i \le 4)$ is no larger than V.

To evaluate the security of PMTEs, Chatterjee et al. [10] used a Support Vector Machine (SVM) to distinguish the real and decoy vaults, and Golla et al. [14] used Kullback-Leibler (KL) divergence. These attacks only exploit the difference between the real and decoy vault distributions but neglect the seeds. We call this type of attack *distribution difference attack*. These attacks cannot exploit the features discussed above. In contrast, our proposed feature attacks only exploit seeds with PMTEs and do not require any knowledge of the real vault distribution. We call this new type of attack *encoding attack*.

3.4 Attacks Against Genomic Data PMTEs

Huang et al. [18] provided a formal proof for the security of their PMTEs. They proved that their PMTEs are indistinguishable from probability models, but did not consider the difference between the real message distribution and probability models. This means their PMTEs resist encoding attacks but have not been evaluated by distribution difference attacks. Although Huang et al. evaluated six probability models with chi-square goodness-of-fit tests, they did not study the influence of their goodness on the security of PMTEs.

In order to evaluate the security, we propose a simple machine learning algorithm to distinguish the real and decoy data (i.e., SNV sequences). As shown in Algorithm 2, we use a training set to train a principal component analysis (PCA) model and a support vector machine (SVM) with a radial basis function (RBF) kernel, where the training set contains the same number of real and decoy SNV sequences, the real sequences are randomly picked from the real dataset, and the decoy sequences are generated by decoding random seeds with the corresponding PMTEs. Specifically, the PCA model is trained and used to reduce the 1000-dimensional sequences in training set to 10 dimensions, and the SVM is trained with the 10-dimensional sequences and the "real/decoy" labels. To estimate the probability that a test sequence s is real, we first use the trained PCA model to reduce s to 10 dimensions, then resort to the trained SVM to classify the reduced sequence and output the probability of it being real. All parameters of the PCA and the SVM are default except "n_components" as 10. Since the default parameters deliver good performance, we do not adjust them. We denote the SVM-estimated probability of s as $p_{\text{PCA+SVM}}(s)$ and propose a PCA+SVM attack with the weight function $p_{\text{PCA+SVM}}$.

4 Generative Probability Models and Generic Encoding Attacks

In this section, we propose a generic conceptual probability model—*Generative Probability Model (GPM)*—to formalize all the existing probability models. This formalization uncovers the principle of encoding attacks. Based on this principle, we propose two generic encoding attacks—a *weak encoding attack* and a *strong encoding attack*.

4.1 Definition

Simple models (e.g., uniform distribution models) assign every message a probability directly, but other complex models cannot. Most complex models (e.g., PCFG models [42]) design a generative method for messages and assign every message a probability with the generated probability of the message. By assigning probabilities to the generating rules, one can get a probability model for the messages. From this point of view, we give a formal definition of *Generative Probability Model*.

Definition 1. A Generative Probability Model (GPM) *is a 5-tuple* $(\mathcal{M}, \mathcal{R}, \mathcal{RS}, G, P)$*, where* $\mathcal{M}$ *is the message space,* $\mathcal{R}$ *is the set of generating rules,* $\mathcal{RS} \subset \mathcal{R}^*$ *is the set of valid generating sequences, G is the generating function mapping a generating sequence RS in* $\mathcal{RS}$ *to a message M in* $\mathcal{M}$*, and P is the probability density function on* $\mathcal{RS}$*. Here* $\mathcal{M}, \mathcal{R}, \mathcal{RS}$ *are finite sets and G is surjective. Then the GPM gives* $\mathcal{M}$ *a probability distribution by*

$$P(M) = \sum_{RS \in G^{-1}(M)} P(RS). \tag{1}$$

In addition, if G is bijective (i.e., for every message in $\mathcal{M}$*, there is only one generating sequence which can generate it), the GPM is* unambiguous*, and otherwise, it is* ambiguous.

Usually, the probability density function P on $\mathcal{RS}$ is given by the conditional probability distribution as follows:

$$P(RS) = \prod_{i=1}^{n} P(r_i | r_1 r_2 \dots r_{i-1}), \tag{2}$$

where $RS = (r_1, r_2, \dots, r_n)$. The conditional probability $P(r_i | r_1 r_2 \dots r_{i-1})$ is usually given in a simple form. Note that the generating sequences in $\mathcal{RS}$ have variable lengths, therefore, the above equation requires that $\mathcal{RS}$ is *prefix-free*, i.e., no sequence in $\mathcal{RS}$ is a prefix of another sequence. Otherwise, the function P defined by Equation 2 is not a probability density function on $\mathcal{RS}$, because $\sum_{RS \in \mathcal{RS}} P(RS) > 1$. Fortunately, if $\mathcal{RS}$ is not prefix-free, it can easily be converted to a prefix-free sequence space $\mathcal{RS}'$ by two simple methods: 1) add a special rule at the beginning of the sequence to represent the length of the sequence; 2) add a special rule at the end of the sequence to represent the end of the sequence. Therefore, without loss of generality, we assume generating sequence spaces of GPMs are all prefix-free.

4.2 Formalization of Existing Models

For a Markov model of order n, a generating rule is a character, and a valid generating sequence is a string. The conditional probability of a rule only depends on last n rules, formally

$$P(a_i | a_1 a_2 \dots a_{i-1}) = P(a_i | a_{i-n} a_{i-n+1} \dots a_{i-1}),$$

where $i > n$ and $P(a_i | a_{i-n} a_{i-n+1} \dots a_{i-1})$ is trained on a training set (RockYou for password vault schemes). The Markov model with distribution-based normalization adds some extra rules $\{L = l\}_{l=1}^{l_{\max}}$ to $\mathcal{R}$, $L = l$ represents that the password length is equal to l, where $1 \le l \le l_{\max}$ and $l_{\max}$ is the max password length (e.g., 30). A valid generating sequence has the form $(L = l, a_1, a_2, \dots, a_l)$ which means generating the length first and then generating the characters. $P(L = l, a_1, a_2, \dots, a_l) = P(L = l) P(a_1, a_2, \dots, a_l)$, where $P(a_1, a_2, \dots, a_l)$ can be calculated as the ordinary Markov model and $P(L = l)$ represents the probability that the length of a password is l. Note that $l_{\max} < \infty$, because the message space $\mathcal{M}$ is finite (the seed space $\mathcal{S}$ is finite).

For a PCFG model, a generating rule is a production rule of the PCFG, a valid generating sequence is a leftmost derivation of a string. The conditional probability of a rule does not depend on any previous rule, formally

$$P(r_i | r_1 r_2 \dots r_{i-1}) = P(r_i),$$

where $P(r_i)$ is also trained on a training set.

For the Golla et al.'s model [14] of password vaults, a generating rule is a character or a value of $|V_i|$ for $0 \le i \le 4$. A valid generating sequence of a vault consists of the following rules: 1) characters of the base password, 2) $|V_i|$, 3) modified characters of passwords in V_i and 4) characters of passwords in V_5. In this case, the conditional probabilities of characters are calculated as the Markov model and $|V_i|$ is calculated by normal distributions.

For the Chatterjee et al.'s model [10] of password vaults, a generating rule is a production rule of the PCFG or a number of production rules with a certain lefthand-side in a vault, a valid generating sequence contains a generating sequence of a sub-grammar and leftmost derivations of passwords based on the sub-grammar. More specifically, a valid generating sequence of the sub-grammar $\{\mathtt{S} \to \mathtt{D}, \mathtt{S} \to \mathtt{W}, \mathtt{D} \to 123456, \mathtt{W} \to \text{password}\}$ is $(\#\mathtt{S} = 2, \mathtt{S} \to \mathtt{D}, \mathtt{S} \to \mathtt{W}, \#\mathtt{D} = 1, \mathtt{D} \to 123456, \#\mathtt{W} = 1, \mathtt{W} \to \text{password})$. The rule $\#\mathtt{X} = i$ represents that there are i rules with the lefthand-side $\mathtt{X}$ in sub-grammar, it is for the sake of the prefix-free property of $\mathcal{RS}$. The conditional probability of the rule $\#\mathtt{X} = i$ only depends on the rule itself, denoted as $P(\#\mathtt{X} = i)$, which is trained on a password vault dataset (Pastebin). The conditional probability of the rule $\mathtt{X} \to str$ is the same as that in PCFG models.

For Huang et al.'s models [18] for genomic data, a generating rule is a character of $\{0, 1, 2\}$ (representing an SNV), a valid generating sequence is a string. The conditional probability of a rule relies on the genomic data model: for the uniform distribution model, it is equal to $\frac{1}{3}$ for each rule; for the public LD model (as discussed above), it depends on the last rule; for Markov model of order n, it depends on the last n rules; for the recombination model, it is calculated by the forward-backward algorithm with a hidden Markov model.

Up to this point, the existing models are all formalized with our proposed GPMs and the distributions of generating sequences are defined by the conditional distributions of generating rules. Beyond that, more probability models can be formalized. For example, neural networks for passwords [27] can be formalized as the same as Markov models except that condition probabilities are calculated by neural networks.

4.3 Generating Graphs

To represent a GPM visually, we propose a *generating graph*, which is a connected directed acyclic graph with a single source and with edges labeled by generating rules. In a generating graph, a generating sequence is illustrated by a path whose edges denote the corresponding generating rules in order. Moreover, a message is figured by a sink (because the generating sequence space is prefix-free) and a path from the source to the sink illustrates one generating sequence of the message. Hence, the path is called a *generating path* of the message. Note that the generating graph of a model is an arborescence, if and only if the model is unambiguous. (Note an arborescence is a directed graph in which there is only one single source and each other vertex has only one directed path from the source.)

As shown in Figure 2, in Chatterjee-PCFG model, there are two generating paths for "password". These two generating paths correspond to two generating sequences: $\{\mathtt{S} \to \mathtt{W}, \mathtt{W} \to$

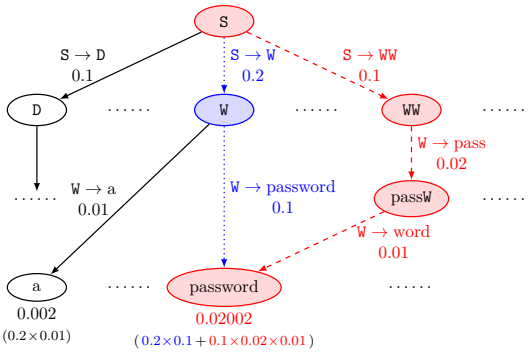


Figure 2: Generating graph of Chatterjee-PCFG

Algorithm 3: The weight function p_{WEA} ($= p_{\text{EC}}$) of the weak encoding attack

```
function p_WEA(S)
    Input: a seed S.
    Output: the weight of S (for sorting in Algorithm 1).
    Obtain the generating sequence RS and the message M of S by
      decoding S
    S' ← encode(M) /* Since encode is a randomized
       algorithm, S' is probably not equal to S.    */
    Obtain the generating sequence RS' of S' by decoding S'
    if RS = RS' then return 1 /* S may be a real seed. */
    else return 0    /* S is definitely a decoy seed. */
end
```

password} and {S → WW, W → pass, W → word}. Further, the probability of the first sequence is $0.2 \times 0.1 = 0.02$ and that of the second one is $0.1 \times 0.02 \times 0.01 = 0.00002$. This makes the probability of "password" be $0.02 + 0.00002 = 0.02002$. Since "password" has two generating sequences, Chatterjee-PCFG model is ambiguous.

4.4 The Principle of Encoding Attacks

The features used by encoding attacks in Section 3 are all based on heuristic analyses of specific PMTEs. Some other features are still neglected due to the lack of a systematic analysis. For example, on Chatterjee et al.'s password vault PMTE [10], the order of rules in the sub-grammar is deterministic for real vaults, but not for decoy seeds. When encoding the vault $V = (123456, \text{password})$, the first two rules in the sub-grammar are S → D, S → W in order. But if the vault V is decoded by a decoy seed, the first two rules may be S → W, S → D in a different order from the real vault.

Fortunately, with the formalizations by GPMs and the visual representations by generating graphs, the principle of encoding attacks is uncovered: *existing PMTEs neglect the ambiguity of GPMs. More specifically, in an ambiguous GPM, there may exist multiple generating paths for a message, but the existing PMTEs only select one deterministic path when encoding.* We name these paths *encoding paths* which can be selected when encoding and meanwhile name these corresponding generating sequences *encoding sequences*. The generating sequence of a seed can be obtained by decoding the seed. Due to the determinacy of encoding paths, encoding attacks can exclude some decoy seeds by checking if the generating path of a seed is an encoding path, without any information of the real message distribution.

We then take Chatterjee et al.'s PMTE [10] for Chatterjee-PCFG as an example. As shown in Figure 2, this PMTE only uses the blue dotted path when encoding "password", but the generating path of a decoy seed may be the red dashed one. In fact, Chatterjee et al. [10] noticed the ambiguity of Chatterjee-PCFG and briefly mentioned that the PMTE needs to choose one parse tree randomly in all parse trees when encoding. However, in Chatterjee et al.'s code [10], they have not implemented the random selection method until now (June 1, 2019) and only one parse tree is selected when encoding. Moreover, Chatterjee et al. [10] completely neglected the ambiguity of the sub-grammar approach. For example, a vault $V = (123456, \text{password})$ is encoded only with the sub-grammar $SG = \{$S → D, S → W, D → 123456, W → password$\}$, but V can be generated by multiple sub-grammars as long as they contain SG. Therefore, the encoding paths definitely have feature UR while other generating paths may not.

Similarly, Golla et al. [14] also did not consider the ambiguity of the reuse-rate approach. For example, $V = (\text{password1}, \text{password1}, \text{password@})$ can be generated by "password1" as the base password with reuse-rates $|V_0| = \frac{2}{3}$ and $|V_1| = \frac{1}{3}$. It also can be generated by "password1" as the base password with reuse-rates $|V_0| = \frac{1}{3}$ and $|V_1| = \frac{2}{3}$. In addition, Golla et al.'s GPMs [14] allow modifying the character of the base password to the same character. Therefore, "password@" may be in V_2 (with "@" modified from "1" and "d" modified from itself). This brings ambiguity to the GPM, i.e., a huge number of generating paths for a vault. Only one deterministic path (the first one for V) is chosen when encoding. Therefore, the encoding paths definitely have feature ED while other generating paths may not.

Any feature utilized by any encoding attack, including features proposed in Section 3.3, the rule-order feature or the base-password feature discussed above, can be seen as a feature of encoding paths.

4.5 Generic Encoding Attacks

Due to the determinacy of encoding paths, we further propose two generic encoding attacks—a *weak encoding attack* and a *strong encoding attack*.

The weak encoding attack is accordance with feature EC (encoding consistency) that the generating path is an encoding path, i.e., the weight function $p_{\text{WEA}} = p_{\text{EC}}$. We use the abbreviation of the attack as the subscript of p for convenience. More specifically, p_{WEA} (i.e., whether a seed S has feature

EC) can be calculated as Algorithm 3.

In contrast to the feature attacks (proposed in Section 3.3) based on some features of encoding path, the weak encoding attack is based on feature EC. Therefore, the seeds having feature EC certainly have other features proposed in Section 3.3. In other words, the weak encoding attack excludes all decoy vaults which are excluded by any feature attack.

As the seeds with feature EC are sorted randomly by the weak encoding attack, we propose a *strong encoding attack* to sort them. Let RS denote the generating sequence of the seed S, then the weight function p_{SEA} is defined as

$$p_{\mathrm{SEA}}(S) = \frac{1}{P(RS)} \times p_{\mathrm{WEA}}(S).$$

4.6 Efficiency of Encoding Attacks

These two generic encoding attacks are efficient for PMTEs with significantly ambiguous GPMs and deterministic encoding paths, such as all existing PMTEs for password vaults. In other words, these attacks recover the encrypted real vaults with a high probability but a small number of online verifications. To make it clear, the weak encoding attack excludes the seeds whose generating paths are not encoding paths, e.g., the red dashed path in Figure 2. Namely, the excluded proportion of the weak encoding attack is equal to the total probability of all generating paths except encoding paths. This means that *the more ambiguous the GPM is, the more efficiency the weak encoding attack can achieve*. As discussed in Section 4.4, in the existing GPMs for password vaults [10, 14], every vault has countless generating paths. Due to the great ambiguity of these GPMs, the weak encoding attack is efficient for the corresponding existing PMTEs with deterministic encoding paths. On the other hand, if a GPM is unambiguous (e.g., the models of genomic data [18]), the PMTE for it can resist encoding attacks naturally. Besides, the strong encoding attack excludes all decoy seeds which are excluded by the weak encoding attacks. Therefore, the strong encoding attack is always more efficient than the weak encoding attack.

5 Probability Model Transforming Encoders

We propose a *generic transforming method* which transforms an arbitrary GPM to a secure PMTE. Further, we give a formal proof that the PMTE transformed by our method is indistinguishable from the GPM.

5.1 Conditional DTEs

Inspired by the way Chatterjee et al.'s PMTEs [10] encoding password character by character or rule by rule, we propose a fundamental concept of PMTE—*conditional distribution transforming encoder (CDTE)*—to encode message rule by rule. A DTE is an encoder transformed from a probability distribution, while a CDTE is an encoder transformed from a conditional probability distribution. Unlike a DTE, a CDTE needs not only the message M but also the condition X to encode M (denoted as $\mathsf{encode}(M|X)$) by the conditional probability distribution $P(\cdot|X)$. It also needs the condition X to decode the seed S (denoted as $\mathsf{decode}(S|X)$). In this aspect, for every condition X, the CDTE $(\mathsf{encode}(\cdot|X), \mathsf{decode}(\cdot|X))$ is a DTE. Interestingly, if the condition X and the message M are mutually independent (i.e., the conditional probability distribution $P(\cdot|X)$ is the same for every condition X), a CDTE degenerates into a DTE. Therefore, we state that DTEs can be seen as a special case of CDTEs. Juels and Ristenpart [21] proposed a generic method to transform a distribution to a DTE and named the DTE *IS-DTE*. For the general conditional distribution, we get a DTE $\mathbf{IS\text{-}DTE}_X$ for each condition X by means of Juels-Ristenpart method and thus we give a general CDTE scheme *IS-CDTE* by the combination $\{\mathbf{IS\text{-}DTE}_X\}_X$.

In the following, we give the details of our IS-CDTE. Let X denote the condition, $\mathcal{X}$ denote the condition space, and $\mathcal{M}_X = \{M_i\}_i$ denote the message space under the condition X. The corresponding conditional probability is $P(M_i|X)$, and the cumulative distribution function is $F_i = \sum_{i'=1}^{i} P(M_{i'}|X)$. When encoding the message M under the condition X, the IS-CDTE randomly generates a real number S in the interval $[F_{i-1}, F_i)$ as a seed of M. When decoding the seed S under condition X, the IS-CDTE searches the interval $[F_{i-1}, F_i)$ containing S and then outputs the corresponding message M_i. Encoding or decoding only requires a binary search of the corresponding CDF (cumulative distribution function) table $\{(M_i, F_i)\}_i$ under the condition. Therefore, the space complexity and the time complexity of the IS-CDTE are $O(|\mathcal{X}| \cdot |\mathcal{M}|)$ and $O(\log(|\mathcal{M}|))$, respectively.

For implementing with encryption, real-number seeds are usually represented as bit strings of length l, i.e., integers in $[0, 2^l)$, where l is a storage overhead parameter. IS-DTEs use the function $\mathsf{round}_l(x)$ converting a real-number seed to an integer seed, where $\mathsf{round}_l(x) = \mathsf{round}(2^l x)$ and round represents rounding function. We use the same method for IS-CDTEs. In such case, the integer seed interval of M_i is $[\mathsf{round}(2^l F_{i-1}), \mathsf{round}(2^l F_i))$. Hence, to ensure that each message has at least one integer seed, l must be greater than or equal to $-\log_2(\min_i P(M_i|X))$. The loss of precision by the discretization with round_l causes a slight difference between these two conditional distributions $\Pr_{\mathbf{IS\text{-}CDTE}}(M|X) = \Pr[M = M' : S \leftarrow_{\$} \mathcal{S}; M' \leftarrow \mathsf{decode}(S|X)]$ and $P(M|X)$, where $\mathbf{IS\text{-}CDTE} = (\mathsf{encode}(\cdot|\cdot), \mathsf{decode}(\cdot|\cdot))$. Fortunately, the difference is negligible in l (see Theorem 4). For convenience, we let $P^{(d)}$ denote the discretization $\Pr_{\mathbf{IS\text{-}CDTE}}$ of P.

5.2 Probability Model Transforming Encoder

Combining IS-CDTEs for the conditional distributions of generating rules, we present a PMTE for the messages, which we call an *IS-PMTE*. Let l denote the storage overhead parameter,

then the IS-PMTE encodes the message M as follows:

1. Parse M and get all generating sequences $G^{-1}(M)$.
2. Calculate the probability $P^{(d)}(RS)$ for each generating sequence RS in $G^{-1}(M)$, where $P^{(d)}(r_1 r_2 \dots r_n) = \prod_{i=1}^{n} P^{(d)}(r_i | r_1 r_2 \dots r_{i-1})$ and $P^{(d)}(r_i | r_1 r_2 \dots r_{i-1})$ is the discretization of $P(r_i | r_1 r_2 \dots r_{i-1})$.
3. Choose a generating sequence RS in $G^{-1}(M)$ with the probability $P^{(d)}(RS|M)$, where $P^{(d)}(RS|M) = \frac{P^{(d)}(RS)}{\sum_{RS' \in G^{-1}(M)} P^{(d)}(RS')}$.
4. Encode each rule r_i in $RS = (r_i)_i$ by the IS-CDTE $\mathsf{encode}(\cdot | r_1 r_2 \dots r_{i-1})$ to a l-bit string S_i.
5. Concatenate $(S_i)_i$, pad the concatenation to a string S of length $ln_{\max}$ with random bits and then output S as a seed for M, where $n_{\max}$ is the maximum length of generating sequences in $\mathcal{RS}$ (i.e., the depth of the generating graph).

In opposite, the IS-PMTE decodes the seed S as follows:

1. Split S into $n_{\max}$ l-bit strings $(S_i)_{i=1}^{n_{\max}}$.
2. Decode S_i to the rule r_i by $\mathsf{decode}(\cdot | r_1 r_2 \dots r_{i-1})$ in turn and ignore the padding bits.
3. Generate the message M from the generating sequence $RS = (r_i)_{i=1}^{n}$ by $M = G(RS)$, then output M as the message of S.

Note that generating sequences vary in length. Because seeds in $\mathcal{S}$ are of fixed length, padding is necessary for some sequences when encoding. Furthermore, as the sequence space $\mathcal{RS}$ is prefix-free, padding bits can be ignored unambiguously when decoding. In addition, note that in Step 2) of encoding the probabilities of sequences are calculated as the discretization $P^{(d)}$ of P, which is necessary to guarantee the uniformity of seeds (see Theorem 3).

Due to the generality of GPMs, IS-PMTEs not only apply to probability models discussed in this paper, but also apply to general probability models, such as neural networks for passwords [27].

Figure 3 depicts how "password" is encoded by our IS-PMTE for the Chatterjee-PCFG model. First, parse all generating sequences of "password". Corresponding to Figure 2, "password" has two generating sequences $\{\mathtt{S} \to \mathtt{W}, \mathtt{W} \to \text{password}\}$ and $\{\mathtt{S} \to \mathtt{WW}, \mathtt{W} \to \text{pass}, \mathtt{W} \to \text{word}\}$. Second, choose a sequence with the probability ($0.02/0.02002 \approx 0.999$ for the first one and 0.001 for the second one). Here we take the second one as an example. Third, encode each generating rule in the sequence by searching the CDF table and translate real-number seeds to bit-string seeds with round_l. Note that in the PCFG models, the conditional probabilities of generating rules do not depend on the previous rules and the rules with the same lefthand-side have the same CDF table. Therefore, the same CDF table is searched for generating rules $\mathtt{W} \to$ pass and $\mathtt{W} \to$ word. Finally, concatenate seeds of rules, pad the concatenation to a fixed length with random bits and get a seed for "password".

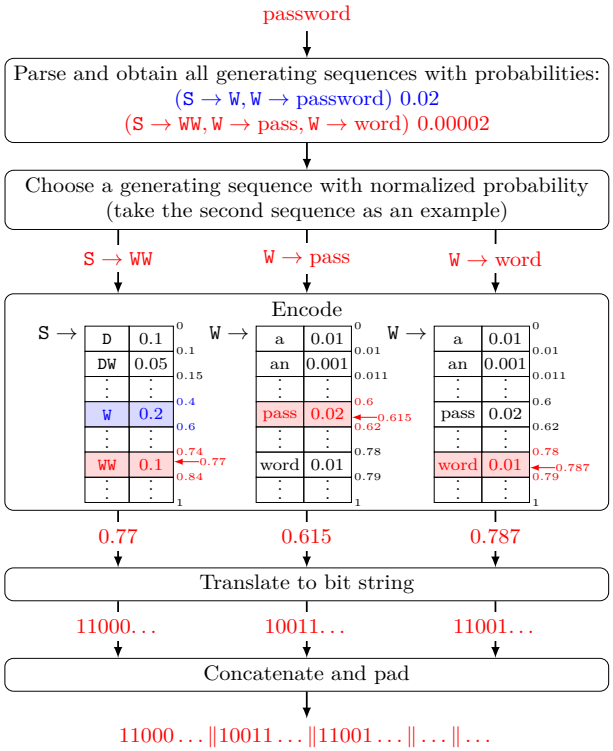


Figure 3: Encode "password" by our IS-PMTE for the Chatterjee-PCFG model

5.3 Difference Between IS-PMTEs and Existing PMTEs

It is easy to get IS-PMTEs from existing GPMs of password vaults and genomic data by our proposed generic transforming method. The following are the differences between the IS-PMTEs and the existing PMTEs [10, 14] for password vaults:

1. IS-PMTEs randomly choose a generating sequence, while the existing PMTEs only choose a deterministic generating sequence. This is the key to resist encoding attacks. Note that the random selection may have high time complexity, fortunately there is a method to reduce it. We leave the details in Appendix C.
2. IS-PMTEs use $\mathsf{round}_l(x)$ to convert a real-number seed to an integer seed, while Chatterjee et al. [10] designed another method to convert a rational-number seed to an integer seed. Unfortunately, Chatterjee et al.'s method cannot be applied to some distributions, e.g., normal distribution. This is because probabilities may be irrational numbers. The method we use (proposed by Juels and Ristenpart [21]) is applicable to arbitrary distributions.

In addition, IS-PMTEs have the same form as the existing PMTEs for password vaults, which is *encode-then-concatenate*. At the same time, the existing PMTEs [18] for genomic data use another *shrink-then-encode* form. When

encoding a string, these genomic data PMTEs shrink the seed interval for each character in the string and further pick a random seed in the final seed interval as the seed for the string. Unfortunately, each interval-shrinking needs to complete large integer arithmetic of length ln to calculate the interval boundary, where l is the storage overhead parameter, and n is the length of the string. This arithmetic costs $\Omega(ln)$ time for each character and $\Omega(ln^2)$ time for the string. In contrast, our IS-PMTEs only need to do integer arithmetic of length l for each character with lower time complexity $\Theta(ln)$ for a string.

5.4 Security of IS-PMTEs

The weak and strong encoding attacks have more generic forms for the PMTEs such as IS-PMTEs who may randomly choose a generating path when encoding. If the PMTE chooses a deterministic generating path when encoding, these generic forms will degenerate to the given forms in Section 4.5. For the weak encoding attack, the more generic form of feature EC is

$$S \in \mathsf{encode}(\mathsf{decode}(S)),$$

where $\mathsf{encode}(M)$ represents all encoded seeds from M. If the seed S does not have feature EC, then S can be decoded to the message $M = \mathsf{decode}(S)$ but cannot be encoded from the message M. Therefore, S is a decoy seed. In order to resist weak encoding attack, it is necessary to ensure that $\mathsf{encode}(M) = \mathsf{decode}^{-1}(M)$ for every message $M \in \mathcal{M}$, where $\mathsf{decode}^{-1}(M)$ represents all seeds which can be decoded into M. In PMTEs with deterministic encoding paths, the generating paths for all seeds in $\mathsf{encode}(M)$ are the same one. In this case, the weak encoding attack degenerates to the given form in Section 4.5.

For the strong encoding attack, the more generic form of the weight function is

$$\Pr_{\mathsf{encode}}(S|\mathsf{decode}(S)),$$

where $\Pr_{\mathsf{encode}}(S|M)$ represents the probability that M is encoded as S under the condition of message M. We denote it as $p_{\mathrm{GSEA}}(S)$. In order to resist strong encoding attack, it is necessary to ensure that $\Pr_{\mathsf{encode}}(S|M)$ are equal for every $S \in \mathsf{decode}^{-1}(S)$, i.e., all valid seeds are uniformly chosen when encoding. We call this property *seed uniformity*. Further, if a DTE has this property, attackers cannot get any useful information except the message from a seed (see Theorem 2). This well explains why our IS-PMTEs choose a generating sequence RS in $G^{-1}(M)$ with the probability $P^{(d)}(RS|M)$ when encoding—it precisely guarantees that seeds are uniform (see Theorem 3). In addition, for PMTEs with deterministic encoding path, the strong encoding attack degenerates to the form in Section 4.5, because $p_{\mathrm{GSEA}} \propto p_{\mathrm{SEA}}$. Let M denote the message, $RS = (r_i)_i$ denote the deterministic generating sequence of M, S denote the seed of M, then we have: 1) if $S \in \mathsf{encode}(M)$, $p_{\mathrm{GSEA}}(S) = \frac{1}{|\mathsf{encode}(M)|} = \frac{1}{|\mathsf{encode}(RS)|} = \frac{1}{|\mathcal{S}|P(RS)} = \frac{1}{|\mathcal{S}|}p_{\mathrm{SEA}}(S)$; 2) otherwise, $p_{\mathrm{GSEA}}(S) = 0 = p_{\mathrm{SEA}}(S)$.

In the following, we prove the security of IS-PMTEs, i.e., decoy seeds/messages are indistinguishable from real ones by any adversary. Let $\mathcal{M}$ denote the message space, $\Pr_{\mathrm{real}}$ denote the probability density function of real messages, $\mathcal{S}$ denote the seed space, and $\mathbf{DTE} = (\mathsf{encode}, \mathsf{decode})$ denote the DTE. Juels and Ristenpart [21] used the advantage of an attacker $\mathcal{A}$ who distinguishes between the real and decoy message-seed pairs to evaluate the security of a DTE, where the advantage is $\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{A}) = |\Pr[\mathcal{A}(S,M) = 1 : M \leftarrow_{\Pr_{\mathrm{real}}} \mathcal{M};\ S \leftarrow_{\$} \mathsf{encode}(M)] - \Pr[\mathcal{A}(S,M) = 1 : S \leftarrow_{\$} \mathcal{S};\ M \leftarrow \mathsf{decode}(S)]|$. This advantage can be simplified, if **DTE** has some properties. *Correctness* is the most basic property of a DTE, which means seeds encoded from the message M can be decoded to M correctly for every message M, i.e., $\mathsf{encode}(M) \subseteq \mathsf{decode}^{-1}(M)$ for every $M \in \mathcal{M}$. If **DTE** is correct, attackers can get the message M from the seed S. Therefore, $\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{A})$ can be simplified to the advantage of attacker $\mathcal{B}$, who distinguishes between the real and decoy seeds, where the advantage is $\mathsf{Adv}^{\mathrm{dte},\mathcal{S}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{B}) = |\Pr[\mathcal{B}(S) = 1 : M \leftarrow_{\Pr_{\mathrm{real}}} \mathcal{M};\ S \leftarrow_{\$} \mathsf{encode}(M)] - \Pr[\mathcal{B}(S) = 1 : S \leftarrow_{\$} \mathcal{S}]|$ (see Theorem 1). Moreover, if **DTE** is correct and seed-uniform, $\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{A})$ can be further simplified to the advantage of an attacker $\mathcal{B}$, who distinguishes between the real and decoy messages, where the advantage is $\mathsf{Adv}^{\mathrm{dte},\mathcal{M}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{B}) = |\Pr[\mathcal{B}(M) = 1 : M \leftarrow_{\Pr_{\mathrm{real}}} \mathcal{M}] - \Pr[\mathcal{B}(M) = 1 : S \leftarrow_{\$} \mathcal{S};\ M \leftarrow \mathsf{decode}(S)]|$ (see Theorem 2). The proof details are given in Appendix A.

Theorem 1. *If **DTE** is correct, then for any attacker $\mathcal{A}$, who distinguishes between the real and decoy message-seed pairs, there exists an attacker $\mathcal{B}$ (as follows), who distinguishes between the real and decoy seeds with* $\mathsf{Adv}^{\mathrm{dte},\mathcal{S}}_{\boldsymbol{DTE},\mathrm{real}}(\mathcal{B}) = \mathsf{Adv}^{\mathrm{dte}}_{\boldsymbol{DTE},\mathrm{real}}(\mathcal{A})$.

$\mathcal{B}(S)$
$M \leftarrow \mathsf{decode}(S)$
return $\mathcal{A}(S,M)$

Theorem 2. *If **DTE** is correct and seed-uniform, for any attacker $\mathcal{A}$, who distinguishes between the real and decoy seeds, there exists an attacker $\mathcal{B}$ (as follows), who distinguishes between the real and decoy messages with* $\mathsf{Adv}^{\mathrm{dte},\mathcal{M}}_{\boldsymbol{DTE},\mathrm{real}}(\mathcal{B}) = \mathsf{Adv}^{\mathrm{dte},\mathcal{S}}_{\boldsymbol{DTE},\mathrm{real}}(\mathcal{A})$.

$\mathcal{B}(M)$
$S \leftarrow_{\$} \mathsf{encode}(M)$
return $\mathcal{A}(S)$

Our proposed IS-PMTEs have the above two properties, thus we neglect the difference between these three types of attackers. Let **GPM** denote the GPM and **IS-PMTE** denote the IS-PMTE of **GPM**. The message generated by **IS-PMTE** (decoding random seed) is indistinguishable from the message generated by **GPM**. Formally, the advantage $\max_{\mathcal{A}} \mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{IS\text{-}PMTE},\mathbf{GPM}}(\mathcal{A})$ is negligible in l (Theorem 5), where $\mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{IS\text{-}PMTE},\mathbf{GPM}}(\mathcal{A}) = |\Pr[\mathcal{A}(M) = 1 : M \leftarrow_{\Pr_{\mathbf{IS\text{-}PMTE}}} \mathcal{M}] - \Pr[\mathcal{A}(M) = 1 : M \leftarrow_{\Pr_{\mathbf{GPM}}} \mathcal{M}]|$, $\Pr_{\mathbf{GPM}}$ is the probability density function P of **GPM** and $\Pr_{\mathbf{IS\text{-}PMTE}}(M) = P^{(d)}(M) = \Pr[M = M' : S \leftarrow_{\$} \mathcal{S}; M' \leftarrow \mathsf{decode}(S)]$. This means that we design a secure PMTE for a GPM. In addition, $\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{IS\text{-}PMTE},\mathrm{real}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{IS\text{-}PMTE},\mathbf{GPM}}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{GPM},\mathrm{real}}(\mathcal{A})$. If **GPM** is an accurate probability model for real messages, i.e., $\mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{GPM},\mathrm{real}}(\mathcal{A})$ is negligible, then $\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{IS\text{-}PMTE},\mathrm{real}}$ is negligible, i.e., **IS-PMTE** is secure for the real message distribution.

Theorem 3. ***IS-PMTE*** *is correct and seed-uniform.*

Theorem 4. ***IS-CDTE*** *is transformed from the conditional probability* $\Pr_{\mathrm{real}}(M|X)$*, the seed length is* l *and* $m = |\mathcal{M}|$*. Then for any condition* X *and any distinguishing attacker* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathrm{dte}}_{\boldsymbol{IS\text{-}CDTE}_X,\mathrm{real}_X}(\mathcal{A}) \leq \frac{m}{2^l}$*, where* $\Pr_{\boldsymbol{IS\text{-}CDTE}_X}(M) = \Pr_{\boldsymbol{IS\text{-}CDTE}}(M|X)$ *and* $\Pr_{\mathrm{real}_X}(M) = \Pr_{\mathrm{real}}(M|X)$*.*

Theorem 5. *Assume the maximum length of generating paths is* n *and each vertex has at most* m *children in the generating graph of* ***GPM****, then* $\mathsf{Adv}^{\mathrm{gpm}}_{\boldsymbol{IS\text{-}PMTE},\boldsymbol{GPM}}(\mathcal{A}) \leq \frac{nm}{2^l}$ *for any attacker* $\mathcal{A}$*. Further,* $\mathsf{Adv}^{\mathrm{dte}}_{\boldsymbol{IS\text{-}PMTE},\mathrm{real}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{gpm}}_{\boldsymbol{GPM},\mathrm{real}}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{gpm}}_{\boldsymbol{IS\text{-}PMTE},\boldsymbol{GPM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{gpm}}_{\boldsymbol{GPM},\mathrm{real}}(\mathcal{A}) + \frac{nm}{2^l}$.

In summary, we propose a generic method for transforming a GPM to a PMTE. The PMTE is secure for the GPM, which means the PMTE is able to resist encoding attacks. To resist distribution difference attacks, an appropriate GPM is needed, for example, statistical language models for natural language texts. Designing such a GPM, however, needs professional knowledge of the real messages, we leave it to experts in related fields.

6 Experimental Results

In this section, we evaluate the security of the existing PMTEs on real datasets under the attacks we propose. In the literature, none of the PMTEs for password vaults can resist encoding attacks as well as none of the PMTEs for genomic data can resist the PCA+SVM attack. But here, we show that our proposed IS-PMTEs for existing password vault models [10, 14] achieve the expected security against encoding attacks as stated in Section 5.4.

6.1 Security Metrics

The ranks of real messages in the order sorted by attackers reflect the security of DTEs. If a DTE is perfectly secure, the real message ranks are evenly distributed under any attack. Accordingly, we use the real message rank distribution as a security metric like [10, 14].

More specifically, we calculate the rank of the message M as follows: 1) generate N decoy messages $\{M_i\}_{i=1}^N$ (by decoding random seeds); 2) calculate the proportion $\hat{r}^-(M)$ (resp. $\hat{r}^+(M)$) of decoy messages with greater (resp. greater or equal) weight than M in $\{M_i\}_{i=1}^N$; 3) pick a random real number in $[\hat{r}^-(M), \hat{r}^+(M)]$ as the rank $\hat{r}(M)$. Same as [10, 14], we set $N = 999$. But different from [10, 14] using average rank $\bar{r}$ (of real messages) and accuracy α (of distinguishing a real message from a decoy one), we use rank cumulative distribution functions (RCDFs) $F(x)$ of real messages to represent attack results. This presentation is more comprehensive than $\bar{r}$ and α. For example, $F^{-1}(1)$ indicates the max rank of real messages, and $F(0)$ indicates the proportion of real messages of rank 0 (i.e., ranking the first). In other words, the attacker excludes $1 - F^{-1}(1)$ proportion decoy messages for all real messages and excludes all decoy messages for $F(0)$ proportion of real messages. In addition, $\bar{r}$ and α can be calculated from $F(x)$ as:

$$\bar{r} = 1 - \int_0^1 F(x)\,dx, \tag{3}$$

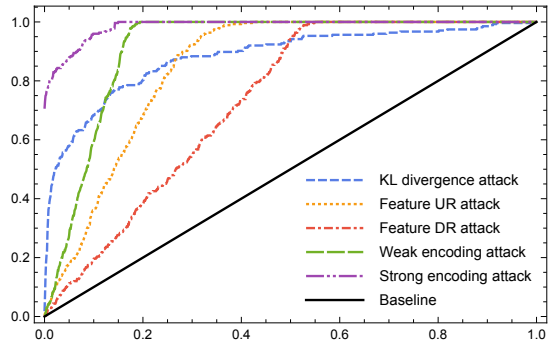
$$\alpha = 1 - \bar{r}. \tag{4}$$

6.2 Datasets

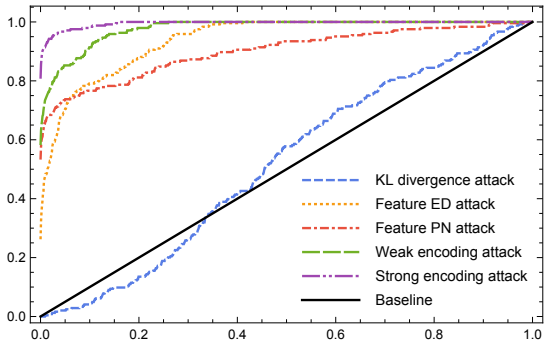
For a fair comparison, we use the same datasets as the previous literature [10, 14, 18]: a password dataset RockYou and a password vault dataset Pastebin for password vault schemes [10, 14], real genomic datasets from HapMap [1] for the genomic data protection scheme [18]. RockYou is a password dataset widely used in password security research, some notable ones like [4, 25, 27, 41], which includes 32.6 million passwords. To the best of our knowledge, Pastebin is the only publicly available dataset for real password vaults so far, and it contains 276 real vaults. Because RockYou and Pastebin are already public and no further harm will be caused, we believe *it is ethical to use them for experiments*. Multiple types of genomic datasets from HapMap are used, including a diploid genotype dataset, a haploid genotype dataset, allele frequency (AF) and linkage disequilibrium (LD) datasets, and recombination rates. The diploid genotype dataset contains 165 persons' SNV sequences. For other details of the above datasets, please refer to [10, 18].

6.3 Evaluating Password Vault PMTEs

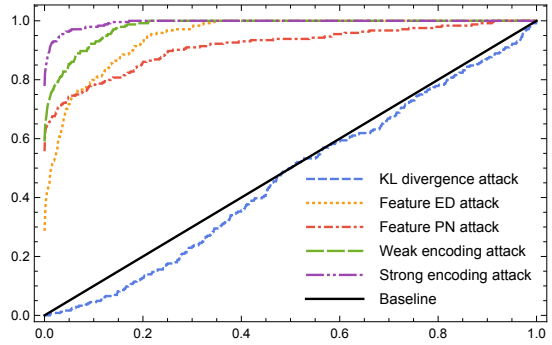
As shown in Figure 4a and Table 1, in Chatterjee et al.'s PMTE [10], the average ranks $\bar{r}$ of real vaults under the feature



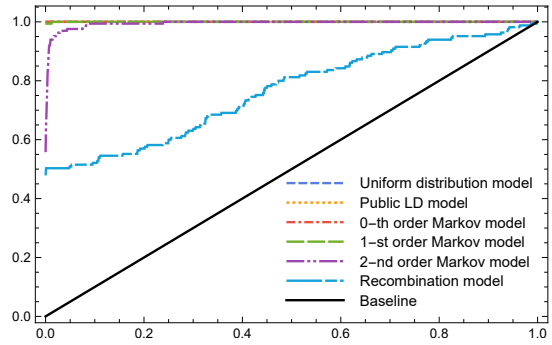
(a) Chatterjee et al.'s PMTE [10]



(b) Golla et al.'s static PMTE [14]



(c) Golla et al.'s adaptive PMTE [14]



(d) Huang et al.'s PMTEs [18] under the PCA+SVM attack

Figure 4: Rank cumulative distribution functions (RCDFs) $F(x)$ of the existing PMTEs

Table 1: The existing PMTEs under encoding attacks or distribution difference attacks

Application	PMTE/Probability model	Attack	$\bar{r}$	α	$F(0)$	$F^{-1}(1)$
Password vault	Chatterjee et al.'s PMTE [10]	KL divergence attack	11.83%	88.17%	1.82%	98.80%
		Feature UR attack	15.14%	84.86%	0.36%	42.24%
		Feature DR attack	26.96%	73.04%	0.00%	54.95%
		Weak encoding attack	8.74%	91.26%	0.36%	19.42%
		Strong encoding attack	1.44%	98.56%	70.55%	15.02%
	Golla et al.'s static PMTE [14]	KL divergence attack	48.26%	51.74%	0.00%	98.70%
		Feature ED attack	6.04%	93.96%	26.23%	41.14%
		Feature PN attack	10.03%	89.97%	53.28%	99.20%
		Weak encoding attack	2.25%	97.75%	58.20%	26.03%
		Strong encoding attack	0.48%	99.52%	80.74%	16.12%
	Golla et al.'s adaptive PMTE [14]	KL divergence attack	53.58%	46.42%	0.00%	100.00%
		Feature ED attack	5.18%	94.82%	28.69%	35.44%
		Feature PN attack	8.60%	91.40%	55.74%	91.79%
		Weak encoding attack	2.01%	97.99%	59.02%	21.22%
		Strong encoding attack	0.58%	99.42%	77.87%	17.22%
Genomic data protection [18]	Uniform distribution model	PCA+SVM attack	0.00%	100.00%	100.00%	0.00%
	Public LD model		0.00%	100.00%	99.39%	0.20%
	0-th order Markov model		0.00%	100.00%	100.00%	0.00%
	1-st order Markov model		0.01%	99.99%	99.39%	1.30%
	2-nd order Markov model		0.53%	99.47%	55.76%	23.92%
	Recombination model		23.46%	76.54%	47.88%	99.90%

UR attack and the feature DR attack are 15.14% and 26.96% respectively, the accuracies α are 84.86% and 73.04%. Moreover, under the feature UR attack, the max rank (i.e., $F^{-1}(1)$) is 42.24%; under the feature DR attack, this number is 54.95%. This means the feature UR attack can exclude at least 57.76% (i.e., $1-F^{-1}(1)$) decoy vaults for every real vault and the feature DR attack can exclude at least 45.05%. Figures 4b, 4c and Table 1 show the performance of Golla et al.'s static PMTE and adaptive PMTE [14], the average ranks under the feature ED attack are 6.04% and 5.18%, while under the feature PN attack are 10.03% and 8.60%. Further, in Golla et al.'s static PMTE, the feature ED attack excludes all decoy vaults for 26.23% (i.e., $F(0)$) real vaults and meanwhile, it excludes at least 58.86% decoy vaults for each real vault. $F(0)$ and $1-F^{-1}(1)$ under the feature PN attack are 53.28% and 0.8% respectively. In Golla et al.'s adaptive PMTE, these numbers are 28.69%, 64.56% under the feature ED attack, and 55.74%, 8.21% under the feature PN attack.

Compared to the above feature attacks, the weak encoding attack has a significant improvement, where the average ranks $\bar{r}$ of Chatterjee et al.'s PMTE [10] and Golla et al.'s (static and adaptive) PMTEs [14] are 8.74%, 2.25%, and 2.01% respectively. The excluded proportions $1-F^{-1}(1)$ are 80.58%, 78.78%, and 73.97%. The strong encoding attack has a further significant improvement compared to the weak encoding attack. The average ranks $\bar{r}$ of these three PMTEs are 1.44%, 0.48%, and 0.57% respectively, which decrease by 84.99%, 83.88%, and 82.78% . Excluded proportions $1-F^{-1}(1)$ are 84.99%, 83.88%, and 82.78% respectively, which also slightly increase by 5.47%, 13.40%, and 5.08%.

Because the KL divergence attack performs better than SVM attacks on all existing PMTEs for password vaults [14], we use it for comparison. As shown in Figures 4a, 4b, 4c and Table 1, the KL divergence attack performs well on the Chatterjee et al.'s PMTE [10], achieving 88.17% accuracy, but it performs almost the same as the randomly guessing attack on Golla et al.'s PMTEs [14], only achieving 46.42%–51.74% accuracy. Further, the RCDFs on Golla et al.'s PMTEs under the KL divergence attack are close to the baseline (the RCDFs under the randomly guessing attack).

For all the existing PMTEs, the curves of RCDFs under the strong encoding attack are all above those under the KL divergence attack. This means that every metric in Table 1 under the strong encoding attack is better than that of the KL divergence attack. More specifically, the average ranks of these three PMTEs under the KL divergence attack are 11.83%, 48.26%, and 53.58%, the accuracies α are 88.17%, 51.74%, and 46.42%. In contrast, the accuracies of the strong encoding attack are 98.56%, 99.52%, and 99.43%, which are 11.78%, 92.35%, and 114.20% higher than those of the KL divergence attack.

In addition, metric values in Table 1 under the KL divergence attack are different from those given in [14], owing to a couple of reasons: 1) for Chatterjee et al.'s PMTE [10], the version of NoCrack used by Golla et al. [14] cannot decode some seeds correctly, therefore have to remedy and reimplement it in the experiments; 2) for Golla et al.'s PMTEs [14], we set the pseudocount of Markov for Laplace smoothing as 1, because under this setting the PMTEs achieve the best security (see Appendix B).

To conclude, the Chatterjee et al.'s PMTE [10] and Golla et al.'s PMTEs [14] are all vulnerable to encoding attacks; meanwhile, Golla et al.'s PMTEs [14] are perfectly secure against the best-reported distribution difference attack.

6.4 Evaluating Genomic Data PMTEs

Different from encoding attacks, the PCA+SVM attack is a distribution difference attack which needs a training set consisting of real and decoy data. We randomly pick 83 individual's SNV sequences in the real dataset[1], generate a decoy sequence for each real sequence, and use them to train our PCA and SVM in the PCA+SVM attack. Then we use remaining 82 individual's sequences in the real dataset and generate N (= 999) decoy sequences for each of them as the test set to compute the RCDF $F(x)$ with the weight function $p_{\text{PCA+SVM}}$. To avoid the impact of randomness on results, we repeat the attack 10 times with different random divisions of the real SNV sequences and newly generated decoy sequences for training/testing, and calculate the average of $F(x)$.

As shown in Figure 4d and Table 1, the PCA+SVM attack achieves more than 99.47% accuracy for all probability models except the recombination model. Even for the recombination model, this attack achieves 76.54% accuracy. This is consistent with Huang et al.'s result [18] that the recombination model performs best. However, it still falls short of the desired security, as our attack excludes all decoy data for 47.88% persons.

To summarize, Huang et al.'s PMTEs for all six models [18] resist encoding attacks but none of them can resist distribution difference attacks. Even the recombination model cannot be rejected at the significance level of 0.2. This means the chi-square goodness-of-fit test is unable to correctly evaluate the security of probability models for generating decoy data.

6.5 Evaluating IS-PMTEs

As stated in Section 5.4, IS-PMTEs resist any encoding attack in theory, we confirm that in practice with IS-PMTEs transformed from existing password vault models. Formally, Theorem 5 demonstrates that the IS-PMTE of an accurate GPM resists arbitrary attacks including encoding attacks. In fact, the IS-PMTE for an arbitrary GPM resists the weak encoding attack. The weight function of the weak encoding attack is constant because every generating path has a chance to be chosen when encoding. This means the weak encoding

[1]We use the small dataset published with the code of GenoGuard on GitHub, which includes 165 persons' SNV sequences of length 1000.

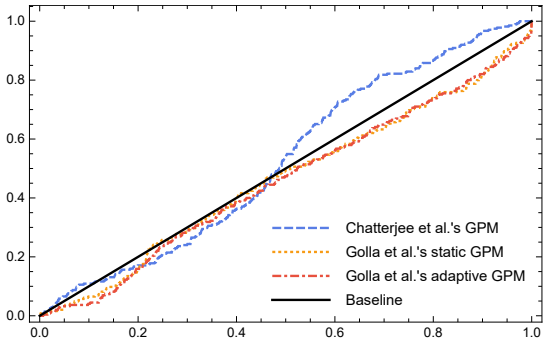


Figure 5: RCDFs of our proposed IS-PMTEs under the strong encoding attack. Note that RCDFs of IS-PMTEs under the weak encoding attack are all equal to the baseline and these under the KL divergence attack are the same as those of the corresponding existing PMTEs in Figure 4.

Table 2: Our IS-PMTEs under the strong encoding attack

Probability model	$\bar{r}$	α	$F(0)$	$F^{-1}(1)$
Chatterjee et al.'s GPM	47.44%	52.56%	0.00%	97.60%
Golla et al.'s static GPM	53.62%	46.38%	0.41%	100.00%
Golla et al.'s adaptive GPM	54.25%	45.75%	0.00%	100.00%

Note: RCDFs of IS-PMTEs under the KL divergence attack are the same as those of existing PMTEs, therefore these metrics under this attack are the same as those in Table 1. RCDFs of IS-PMTEs under the weak encoding attack are the same as those under the randomly guessing attack, therefore these metrics are trivial (50% for $\bar{r}$, 50% for α, 0% for $F(0)$ and 100% for $F^{-1}(1)$).

attack degenerates to the randomly guessing attack (with a constant weight function). In contrast, the weight function p_{GSEA} of the strong encoding attack is inconstant, therefore, RCDFs under the strong encoding attack depend on GPMs.

To evaluate the security of IS-PMTEs for existing vault models under the strong encoding attack, it is necessary to implement the random selection method for generating paths with the parsing function G^{-1}. However, in existing GPMs for password vaults, there are numerous generating paths for messages (as discussed in Section 4.4), therefore, it has high time complexity to parse all generating paths (see the discussion in Appendix C). For example, in Chatterjee et al.'s GPM [10], a vault $V = (123456, \text{password})$ can be generated by any sub-grammar containing $SG = \{\mathtt{S} \to \mathtt{D},\ \mathtt{S} \to \mathtt{W},\ \mathtt{D} \to 123456,\ \mathtt{W} \to \text{password}\}$. It has high time complexity to enumerate all these sub-grammars and calculate the probabilities of generating V by them. Instead, we carry out simulation experiments under the degenerated form of the strong encoding attack with the weight function p_{SEA}. Because all generating paths are encoding paths, there is no seed S with $p_{\text{SEA}}(S) = 0$, i.e., $p_{\text{SEA}}(S) = \frac{1}{P(RS)}$ for every seed S. Accordingly, we use this weight function to sort the seeds in simulation experiments.

Compared to the existing PMTEs, IS-PMTEs transformed from the existing GPMs have a significant improvement on security. As shown in Figure 5 and Table 2, all RCDFs of the IS-PMTEs under the strong encoding attack are approaching to the baseline, i.e., the RCDF under the randomly guessing attack. Average ranks $\bar{r}$ are all near to the expected value of 50%, which are 47.44%, 53.62%, and 54.25%, respectively. Meanwhile, the accuracies are 52.56%, 46.38%, and 45.75%, respectively. Recall that accuracies of existing PMTEs under the strong encoding attacks are 98.56%, 99.52%, and 99.42%, respectively.

Note that our IS-PMTEs have the same decoy message distributions with the corresponding GPMs. This means our IS-PMTEs achieve the same security as the existing PMTEs for the same GPMs under distribution difference attacks. Due to the good performance of Golla et al.'s PMTEs [14] against the best-reported distribution difference attack, our IS-PMTEs for Golla et al.'s GPMs achieve the expected security under both encoding attacks and distribution difference attacks.

7 Conclusion

With encoding attacks and distribution difference attacks, we evaluate three typical existing PMTEs, including two for password vaults and one for genomic data. Using a PCA and an SVM, a distribution difference attack can distinguish real and decoy genomic data with high accuracy. Different from distribution difference attacks exploiting the difference between real and decoy message distributions, encoding attacks are a new type of attack we propose, which exploit the difference between probability models and PMTEs. Encoding attacks can exclude most decoy password vaults/seeds, without any knowledge of real vault distributions.

Further, we introduce a generic conceptual probability model—*generative probability model (GPM)*—to formalize probability models. With the formalization by GPMs, the principle of encoding attacks is uncovered. Based on this principle, we propose two generic and more efficient encoding attacks. In addition, we propose a generic method for transforming an arbitrary GPM to a PMTE. We prove that PMTEs transformed by this method are information-theoretically indistinguishable from the corresponding GPMs, thus can resist encoding attacks. Using this transforming method, we simplify the task of designing a secure PMTE to the task of designing an accurate GPM. Designing such a GPM needs professional knowledge of real messages, we leave it to experts in related fields for future work.

Acknowledgment

The authors are grateful to the anonymous reviewers and the shepherd, Prof. Vincent Bindschaedler, for their invaluable comments that highly improve the completeness of the paper. We also give our special thanks to Prof. Kaitai Liang and

Qianchen Gu for their insightful suggestions and invaluable help. This research was supported by the National Key R&D Program of China under Grant No.2017YFB1200700, and by the National Natural Science Foundation of China (NSFC) under Grant No.61672059.

References

[1] Hapmap. http://hapmap.ncbi.nlm.nih.gov/downloads/index.html.en.

[2] LastPass and YubiKey. https://lastpass.com/yubico/.

[3] Ingolf Becker, Simon Parkin, and M Angela Sasse. The rewards and costs of stronger passwords in a university: linking password lifetime to strength. In *Proc. USENIX Security 2018*, pages 239–253, 2018.

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

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

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

[7] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *Proc. IEEE S&P 2012*, pages 538–552, 2012.

[8] Joseph Bonneau, Cormac Herley, Paul C Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proc. IEEE S&P 2012*, pages 553–567.

[9] Daniel Buschek, Alexander De Luca, and Florian Alt. Improving accuracy, applicability and usability of keystroke biometrics on mobile touchscreen devices. In *Proc. ACM CHI 2015*, pages 1393–1402.

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

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

[12] Warwick Ford and Burton S Kaliski. Server-assisted generation of a strong secret from a password. In *Proc. WETICE 2000*, pages 176–180.

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

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

[15] Maximilian Golla and Markus Dürmuth. On the accuracy of password strength meters. In *Proc. ACM CCS 2018*, pages 1567–1582.

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

[17] Douglas N Hoover and BN Kausik. Software smart cards via cryptographic camouflage. In *Proc. IEEE S&P 1999*, pages 208–215.

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

[19] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. Device-enhanced password protocols with optimal online-offline protection. In *Proc. ACM CCS 2016*, pages 177–188.

[20] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. Two-factor authentication with end-to-end password security. In *Proc. PKC 2018*, pages 431–461. Springer.

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

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

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

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

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

[26] Michelle L Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring password guessability for an entire university. In *Proc. ACM CCS 2013*, pages 173–186.

[27] William Melicher, Blase Ur, Sean M Segreti, Saranga

Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *Proc. USENIX Security 2016*, pages 175–191.

[28] Fabian Monrose, Michael K Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. *Int. J. Netw. Secur.*, 1(2):69–83, 2002.

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

[30] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *Proc. ACM CCS 2017*, pages 295–310.

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

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

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

[34] Richard Shay, Saranga Komanduri, Adam L Durity, Phillip Seyoung Huh, Michelle L Mazurek, Sean M Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Designing password policies for strength and usability. *ACM Trans. Inform. Syst. Secur.*, 18(4):13, 2016.

[35] Maliheh Shirvanian, Stanislaw Jarecki, Nitesh Saxena, and Naveen Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *Proc. NDSS 2014*.

[36] Maliheh Shirvanian, Stanislaw Jarecki, Nitesh Saxena, and Naveen Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *Proc. NDSS 2014*, pages 1–16. The Internet Society.

[37] Maliheh Shirvanian, Stanislaw Jareckiy, Hugo Krawczykz, and Nitesh Saxena. Sphinx: A password store that perfectly hides passwords from itself. In *Proc. ICDCS 2017*, pages 1094–1104.

[38] Joe Siegrist. LastPass security notification, July 2015. https://blog.lastpass.com/2015/06/lastpass-security-notice.html/.

[39] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, et al. Design and evaluation of a data-driven password meter. In *Proc. ACM CHI 2017*, pages 3775–3786.

[40] Ding Wang, Debiao He, Haibo Cheng, and Ping Wang. fuzzypsm: A new password strength meter using fuzzy probabilistic context-free grammars. In *Proc. IEEE DSN 2016*, pages 595–606.

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

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

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

A Proofs in Section 5

Proof of Theorem 1.

$$
\begin{aligned}
&\mathsf{Adv}^{\mathrm{dte},\mathcal{S}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{B})\\
=&|\Pr[\mathcal{B}(S)=1:M\leftarrow_{\Pr_{\mathrm{real}}}\mathcal{M};S\leftarrow_{\$}\mathsf{encode}(M)]\\
&-\Pr[\mathcal{B}(S)=1:S\leftarrow_{\$}\mathcal{S}]|\\
=&|\Pr[\mathcal{A}(S,M')=1:M\leftarrow_{\Pr_{\mathrm{real}}}\mathcal{M};S\leftarrow_{\$}\mathsf{encode}(M);\\
&M'\leftarrow\mathsf{decode}(S)]\\
&-\Pr[\mathcal{A}(S,M')=1:S\leftarrow_{\$}\mathcal{S};M'\leftarrow\mathsf{decode}(S)]|\\
=&|\Pr[\mathcal{A}(S,M)=1:M\leftarrow_{\Pr_{\mathrm{real}}}\mathcal{M};S\leftarrow_{\$}\mathsf{encode}(M)]\\
&-\Pr[\mathcal{A}(S,M)=1:S\leftarrow_{\$}\mathcal{S};M\leftarrow\mathsf{decode}(S)]|\\
=&\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{A}).
\end{aligned}
$$

□

Proof of Theorem 2.

$$
\begin{aligned}
&\mathsf{Adv}^{\mathrm{dte},\mathcal{M}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{B})\\
=&|\Pr[\mathcal{B}(M)=1:M\leftarrow_{\Pr_{\mathrm{real}}}\mathcal{M}]\\
&-\Pr[\mathcal{B}(M)=1:S\leftarrow_{\$}\mathcal{S};M\leftarrow\mathsf{decode}(S)]|\\
=&|\Pr[\mathcal{A}(S')=1:M\leftarrow_{\Pr_{\mathrm{real}}}\mathcal{M};S'\leftarrow_{\$}\mathsf{encode}(M)]\\
&-\Pr[\mathcal{A}(S')=1:S\leftarrow_{\$}\mathcal{S};M\leftarrow\mathsf{decode}(S);\\
&S'\leftarrow_{\$}\mathsf{encode}(M)]|\\
=&|\Pr[\mathcal{A}(S)=1:M\leftarrow_{\Pr_{\mathrm{real}}}\mathcal{M};S\leftarrow_{\$}\mathsf{encode}(M)]\\
&-\Pr[\mathcal{A}(S)=1:S\leftarrow_{\$}\mathcal{S}]|\\
=&\mathsf{Adv}^{\mathrm{dte},\mathcal{S}}_{\mathbf{DTE},\mathrm{real}}(\mathcal{A}).
\end{aligned}
$$

□

Proof of Theorem 3. **IS-DTE** is correct, therefore, the combination $\mathbf{IS\text{-}CDTE}=\{\mathbf{IS\text{-}DTE}_X\}_{X\in\mathcal{X}}$ is correct. In addition, because $\mathcal{RS}$ is prefix-free, the padding bits can be ignored unambiguously when decoding. Thus, **IS-PMTE** is correct.

Let S be a seed of the message M, $RS=(r_i)_{i=1}^{n}$ be the generating sequence of S, then the length of padding bits is $ln_{\max}-ln$ and

$$\Pr_{\mathsf{encode}}(S|M)$$

$$
\begin{aligned}
&= \frac{P^{(d)}(RS)}{P^{(d)}(M)} \cdot \frac{1}{2^{ln_{\max}-ln}} \prod_{i=1}^{n} \frac{1}{|\mathsf{encode}(r_i|r_1 r_2 \dots r_{i-1})|} \\
&= \frac{P^{(d)}(RS)}{P^{(d)}(M)} \cdot \frac{1}{2^{ln_{\max}-ln}} \prod_{i=1}^{n} \frac{1}{2^{l} P^{(d)}(r_i|r_1 r_2 \dots r_{i-1})} \\
&= \frac{P^{(d)}(RS)}{P^{(d)}(M)} \cdot \frac{1}{2^{ln_{\max}}} \prod_{i=1}^{n} \frac{1}{P^{(d)}(r_i|r_1 r_2 \dots r_{i-1})} \\
&= \frac{P^{(d)}(RS)}{P^{(d)}(M)} \cdot \frac{1}{2^{ln_{\max}} P^{(d)}(RS)} \\
&= \frac{1}{2^{ln_{\max}} P^{(d)}(M)}.
\end{aligned}
$$

Therefore, **IS-PMTE** is seed-uniform. □

Proof of Theorem 4. According to the definition of $\mathbf{IS\text{-}CDTE}_X$, $\Pr_{\mathbf{IS\text{-}CDTE}_X}(M_i) = \Pr^{(d)}_{\mathrm{real}_X}(M_i) = \mathsf{round}_l(F_i) - \mathsf{round}_l(F_{i-1})$ and $\Pr_{\mathrm{real}_X}(M_i) = F_i - F_{i-1}$, so that $|\Pr_{\mathbf{IS\text{-}CDTE}_X}(M_i) - \Pr_{\mathrm{real}_X}(M_i)| \le \frac{1}{2^l}$. To summarize, $\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{IS\text{-}CDTE}_X,\mathrm{real}_X}(\mathcal{A}) \le \sum_{M\in\mathcal{M}} |\Pr_{\mathbf{IS\text{-}CDTE}_X}(M) - \Pr_{\mathrm{real}_X}(M)| \le \frac{m}{2^l}$. □

Proof of Theorem 5. $\Pr_{\mathbf{IS\text{-}PMTE}}$ is the discretization of $\Pr_{\mathbf{GPM}}$. Similarly, discretizing the first i levels of the generating graph (and keeping the rest levels unchanged) gets a GPM, denoted as $\mathbf{GPM}_i$. Therefore, $\Pr_{\mathbf{GPM}_i}(r_j|r_1 r_2 \dots r_{j-1}) = \Pr_{\mathbf{GPM}_{i-1}}(r_j|r_1 r_2 \dots r_{j-1})$ for $j \neq i$ and by Theorem 4 $|\Pr_{\mathbf{GPM}_i}(r_i|r_1 r_2 \dots r_{i-1}) - \Pr_{\mathbf{GPM}_{i-1}}(r_i|r_1 r_2 \dots r_{i-1})| \le \frac{1}{2^l}$, then

$$
\begin{aligned}
&\mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{GPM}_i,\mathbf{GPM}_{i-1}}(\mathcal{A}) \\
&\le \sum_{M\in\mathcal{M}} |\Pr_{\mathbf{GPM}_i}(M) - \Pr_{\mathbf{GPM}_{i-1}}(M)| \\
&\le \sum_{RS\in\mathcal{RS}} |\Pr_{\mathbf{GPM}_i}(RS) - \Pr_{\mathbf{GPM}_{i-1}}(RS)| \\
&= \sum_{(r_j)_j\in\mathcal{RS}} \Big|\prod_j \Pr_{\mathbf{GPM}_i}(r_j|r_1 r_2 \dots r_{j-1}) \\
&\qquad\qquad - \prod_j \Pr_{\mathbf{GPM}_{i-1}}(r_j|r_1 r_2 \dots r_{j-1})\Big| \\
&= \sum_{(r_j)_j\in\mathcal{RS}} \prod_{j\neq i} \Pr_{\mathbf{GPM}_i}(r_j|r_1 r_2 \dots r_{j-1}) \times \\
&\quad |\Pr_{\mathbf{GPM}_i}(r_i|r_1 r_2 \dots r_{i-1}) - \Pr_{\mathbf{GPM}_{i-1}}(r_i|r_1 r_2 \dots r_{i-1})| \\
&\le \sum_{(r_j)_j\in\mathcal{RS}} \prod_{j\neq i} \Pr_{\mathbf{GPM}_i}(r_j|r_1 r_2 \dots r_{j-1}) \frac{1}{2^l} \\
&= \frac{m}{2^l}.
\end{aligned}
$$

Because $\Pr_{\mathbf{GPM}_0} = \Pr_{\mathbf{GPM}}$ and $\Pr_{\mathbf{GPM}_n} = \Pr_{\mathbf{IS\text{-}PMTE}}$, $\mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{IS\text{-}PMTE},\mathbf{GPM}}(\mathcal{A}) \le \sum_{i=1}^{n} \mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{GPM}_i,\mathbf{GPM}_{i-1}}(\mathcal{A}) \le \frac{nm}{2^l}$. Moreover, $\mathsf{Adv}^{\mathrm{dte}}_{\mathbf{IS\text{-}PMTE},\mathrm{real}}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{IS\text{-}PMTE},\mathbf{GPM}}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{GPM},\mathrm{real}}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{gpm}}_{\mathbf{GPM},\mathrm{real}}(\mathcal{A}) + \frac{nm}{2^l}$. □

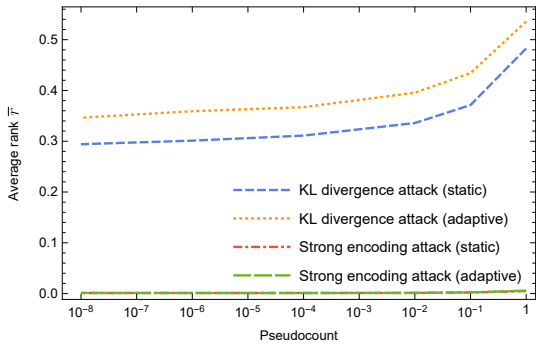


Figure 6: Average rank vs. pseudocount for Golla et al.'s PMTEs [14]

B The Security of Golla et al.'s PMTEs [14] with Different Pseudocounts

We find out that pseudocounts (smoothing parameter) of Markov models with Laplace smoothing in Golla et al.'s PMTEs [14] have a significant influence on the security of PMTEs. As shown in Figure 6 and Table 3, the average rank $\bar{r}$ increases as pseudocount increases under both the KL divergence attack and the strong encoding attacks, and meanwhile, the accuracy α decreases. This means that Golla et al.'s PMTEs [14] achieve the best security when pseudocount is 1. Other metrics in Table 3 also support this conclusion.

Further, when pseudocount is 1, α of the KL divergence attacks are 51.74% and 46.42% on Golla et al.'s static and adaptive PMTEs, respectively. This means Golla et al.'s PMTEs/GPMs almost achieve the expected security ($\alpha = 50\%$) under the best-reported distribution difference attacks.

C The Complexity of IS-PMTEs and Optimization for Encoding

The complexity of an IS-PMTE is of the same order as that of the corresponding GPM. The IS-PMTE stores the CDF table as well as the GPM stores the PDF (probability density function) table. These two tables are of the same size, which means the PMTE and the GPM have the same order of space complexity. When encoding a message, the IS-PMTE needs to obtain all generating sequences for the message and calculate the probability of each sequence, which also needs to be done when the GPM calculates the (total) probability of the message. Moreover, encoding/decoding a sequence needs to do binary search on CDF tables, and meanwhile, calculating the probability of the sequence needs to do binary search on PDF tables. Therefore, the IS-PMTE and the GPM have the same order of time complexity (for encoding messages and calculating message probabilities, respectively).

However, it suffers from high time complexity to obtain all generating paths for some GPMs with great ambiguity. As

Table 3: Golla et al.'s PMTEs [14] with different pseudocounts

Pseudo-count	Attack	Golla et al.'s static PMTE [14]				Golla et al.'s adaptive PMTE [14]			
		$\bar{r}$	α	$F(0)$	$F^{-1}(1)$	$\bar{r}$	α	$F(0)$	$F^{-1}(1)$
1	KL divergence attack	48.26%	51.74%	0.00%	98.70%	53.58%	46.42%	0.00%	100.00%
10^{-1}		37.13%	62.87%	0.00%	99.50%	43.42%	56.58%	0.00%	100.00%
10^{-2}		33.59%	66.41%	2.46%	99.40%	39.55%	60.45%	2.87%	100.00%
10^{-4}		31.11%	68.89%	11.89%	99.20%	36.71%	63.29%	11.89%	100.00%
10^{-6}		30.11%	69.89%	14.75%	99.00%	35.91%	64.09%	14.75%	100.00%
10^{-8}		29.42%	70.58%	17.21%	99.50%	34.62%	65.38%	16.80%	100.00%
1	Strong encoding attack	0.48%	99.52%	80.74%	16.12%	0.58%	99.42%	77.87%	17.22%
10^{-1}		0.22%	99.78%	87.30%	10.11%	0.23%	99.77%	82.38%	9.81%
10^{-2}		0.12%	99.88%	90.57%	9.51%	0.14%	99.86%	90.16%	7.81%
10^{-4}		0.12%	99.88%	92.21%	10.51%	0.10%	99.90%	92.21%	6.41%
10^{-6}		0.11%	99.89%	91.80%	7.91%	0.11%	99.89%	90.98%	8.41%
10^{-8}		0.11%	99.89%	91.80%	8.61%	0.13%	99.87%	92.62%	9.51%

discussed in Section 4.4, in Chatterjee et al.'s GPM [10], a vault can be generated by numerous sub-grammars in Chatterjee et al.'s GPMs. In Golla et al.'s [14] GPMs, a vault can be generated by different base passwords, different cardinalities of subsets and different modified characters. Fortunately, some generating paths can be pruned to reduce the time complexity of encoding. In some models, the dependency of some rules is ignored (by assuming the rules are independent). This triggers some unnecessary paths which can be pruned. For example, in Golla et al.'s GPMs [14], the modified character b_i of passwords in V_i ($1 \leq i \leq 4$) and the corresponding character a_i of the base password are assumed to be independent. In other words, the character of the base password can be modified to itself, i.e., $a_i = b_i$. This yields significant ambiguity. By prohibiting this, we can prune the branch of the original character a_i when generating the modified character b_i. More specifically, the steps of the pruned encoding are as follows: 1) copy a CDF table and delete a_i in the new table; 2) renormalize remaining characters; 3) encode b_i through the renormalized CDF table; 4) abandon the copied CDF table (use the original table for encoding other characters). From the view of the generating graph, the branch of a_i on the node of generating b_i are pruned, resulting in a decrease of time complexity. Besides, the following branches can also be pruned: 1) the character of passwords in V_5 which is the same as the corresponding character of the base password; 2) the cardinality of V_i which is larger than the number of rest passwords. By pruning unnecessary branches on some nodes in the generating graph, we greatly reduce the ambiguity of Golla et al.'s GPMs [14]. For the vaults V, there are only n' generating paths left, where n' is the number of unique passwords in V. Each path corresponds to a different password for generating the vault as the base password.

In Chatterjee et al.'s GPM [10], some unnecessary branches can also be pruned efficiently, e.g., the branches of duplicate rules. However, the branches of unused rules are difficult to be pruned. For example, a vault V of size 2 is generated by the sub-grammar $SG = \{\texttt{S} \to \texttt{D},\ \texttt{S} \to \texttt{W},\ \texttt{D} \to 123456,\ \texttt{W} \to \text{password}\}$. If the first password in V is "123456", then the second one must be "password" to avoid unused rules, i.e., the branch of the rule $\texttt{S} \to \texttt{D}$ should be pruned when generating the second password. In addition, some sub-grammars cannot generate a vault of size 2 without unused rules, for example, the sub-grammars consist of three rules with the lefthand-side S. It also needs to be pruned the branches of all these sub-grammars and renormalize the rest branches. Therefore, in order to prune the branches of unused rules, it is necessary to prune and renormalize branches on almost all nodes in the generating graph. This pruning is difficult because of the high time complexity, especially for the vaults of large sizes. Another simple and straightforward method is to add extra rules in the sub-grammar randomly when encoding. It seems to address this problem. However, the Chatterjee et al.'s GPM [10] with this rule-adding method resists the weak encoding attack but still suffers from the strong encoding attack unless the probability of adding extra rules is equal to the probability of the generating path. This is because the DTE must be seed-uniform in order to resist the strong encoding attack. Moreover, calculating the probability of adding extra rules has the same order of time complexity as calculating the probability of the generating path. Therefore, if this rule-adding method guarantees the property of seed-uniformity, it is equivalent to our method which randomly chooses a generating path with its probability. In other words, the rule-adding method does not perform efficiently in resisting the strong encoding attack. To conclude, we state that a secure DTE of the sub-grammar approach does have high time complexity.

To get rid of the high time complexity of encoding sub-grammars, we propose a design principle for GPMs—*minimizing the ambiguity of the GPM*—to reduce the time complexity of encoding in the corresponding PMTEs. Instead of optimizing the encoding algorithm after designing a GPM with great ambiguity, it may be better to minimize the ambiguity when designing the GPM.