An Enhancement of Honey Encryption Algorithm Applied in Online Patient Portals

Ryan James Agustin, Jhack Rowlan Concha

College of Information Systems and Technology, Pamantasan ng Lungsod ng Maynila – Philippines

DOI: https://dx.doi.org/10.47772/IJRISS.2025.9010008

Received: 30 December 2024; Accepted: 02 January 2025; Published: 27 January 2025

ABSTRACT

The Honey Encryption Algorithm is a novel encryption scheme designed to encrypt messages using low-entropy keys, such as passwords. When an incorrect key is used for decryption, it returns a honeyword—a fictitious message intended to mislead attackers into thinking they have accessed the correct data. However, in password-based authentication systems, a usability issue arises when legitimate users accidentally input their password in a way that results in a honeyword. This study addresses the problem of typo-safety in Honey Encryption Algorithm by employing Long Short-Term Memory (LSTM) models combined with Damerau-Levenshtein distance metric to generate honeywords that are less likely to be typed by legitimate users, while still being effective against adversarial attacks. The Enhanced Honey Encryption Algorithm achieved a typo rate of 29%, with only 4.57% of typos producing a honeyword candidate. In comparison, the original Honey Encryption Algorithm using a Take-a-Tail method had a 71% typo rate, with 27.71% of typos resulting in honeyword candidates. A password survey was conducted on 100 participants, asking them to re-enter 20 passwords under timed and environmental constraints for ecological validity. Both algorithms were tested using honeyword sets generated from a set of passwords sourced from the phpBB password list. The Enhanced Honey Encryption Algorithm's 29% typo rate and 4.57% honeyword rate indicate a lower probability of legitimate users' typos resulting in a honeyword, resulting in a significant improvement in usability in password-based authentication systems.

Keywords: Honey Encryption, Password, Security, Deception, Machine Learning

INTRODUCTION

Personally Identifiable Information refers to any information that can be used to trace or distinguish an individual's identity such as their medical, financial, educational and employment information (McCallister, 2018). Personal Health Information is a subset of PII, which refers to any information in the medical record or designated record set that can be used to identify an individual and that was created, used, or disclosed in the course of providing a health care service such as diagnosis or treatment (UC Berkeley, n.d.).

Digitization of a patient's personal health information may come in two forms: Electronic Medical Record and Electronic Health Record. The difference is that EMRs only include personal health information sourced from a singular healthcare provider such as a clinic and are limited in that they don't travel outside the practice easily, while EHRs are broader and more holistic in that they encompass all health information sourced from different healthcare entities and settings, thus is usually maintained by all clinicians involved in the patient's care, and allows easy sharing of medical information with the stakeholders, enabling a scenario where a patient's information would follow him or her through the various modalities of care engaged by that individual across different medical settings (Garrett & Seidman, 2011).

Patients can access their personal health information, anytime and anywhere through an Online Patient Portal. Patient Portals are defined as secure websites that enable patients a convenient way to access their personal health information and a variety of medical information and features 24/7 just by using their username, and password as login credentials and an internet connection (HealthIT.gov, 2017).

Today, Online Patient Portals are commonplace. The features offered vary from one Online Patient Portal to another. In general, users can access their personal health information from all their healthcare providers 24 hours a day, in addition to this, they can make payments, refill prescriptions, update insurance & contact information, and view their medical history, test results (MedlinePlus Medical Encyclopedia, 2022).

As mentioned, Online Patient Portals typically authenticate their users using login credentials consisting of their username, and password. This is not a problem with Online Patient Portals that are hosted locally - only accessible within the premises of the health organization. With those that are hosted on the internet and accessible by anyone, however, this presents a genuine concern. There are a multitude of ways on how to crack a password, from dictionary attacks, brute-force attacks, rainbow table attacks, and so on. Verizon in its 2017 Data Breach Investigations Report (DBIR) states that an estimate of 81% of current data breaches are attributed to hacked, stolen or weak passwords (Verizon, 2017).

The Honey Encryption Algorithm is an encryption scheme, a novel approach first introduced in Juels and Ristenpart's 2014 paper "Honey Encryption: Security Beyond the Brute-Force Bound". It constitutes encrypting messages using a low min-entropy key like passwords, and producing a ciphertext from it. When the ciphertext's decryption is attempted using any incorrect key, it produces honey messages, which is a plausible-looking but fake message and alerts the system administrators. HE, is then, a decoy-based encryption scheme that yields plausible looking plaintexts when decrypted using incorrect keys (honeywords) leading the adversary to believe that he is in possession of the plaintext when in actuality, what he has is the honey message (Omolara et al., 2019a). This way, the attacker is deceived into thinking they have accessed the system, and is viewing the genuine, actual information of the target. Honey Encryption is most commonly used to provide brute-force security in low entropy environments where brute-force attacks are effective, but can also be used to provide additional security to an authentication system as in Omolara et al.'s (2020b) HoneyDetails where honeywords is used in conjunction with the honey encryption algorithm scheme.

That said, the Honey Encryption Algorithm, through its Honey-based Authentication component as extended by Omolara et al. (2020b), is currently plagued by a prominent usability issue in the form of user typos of the password that may result in a sweet word.

The goal of this research is to enhance the Honey Encryption Algorithm by addressing the production of inaccurate data results due to a usability issue that is present during user password entry. Then, use the enhanced algorithm in an Online Patient Portal to better secure users' Personal Health Information, and prevent the disclosure of users' sensitive data to unauthorized parties.

LITERATURE REVIEW

Honey Encryption Scheme

The Honey Encryption Algorithm is an encryption scheme, a novel approach first introduced in Juels and Ristenpart's (2014) paper "*Honey Encryption: Security Beyond the Brute-Force Bound*". It constitutes encrypting messages using a low min-entropy key like passwords, and producing a ciphertext from it. When the ciphertext's decryption is attempted with any incorrect key, it produces honey messages, which are a plausible-looking but fake message which are displayed to the attacker. Simultaneously alerting the system administrators in the process also. This way, the attacker is deceived into thinking they have accessed the system, and is viewing the genuine, actual information of the target while the administrators are aware of the attackers' intrusion. Honey Encryption is most used to provide brute-force security in low entropy environments such as Social Security Numbers, and PINS where brute-force attacks are effective because of their structured nature.

The Honey Encryption Scheme also has other applications other than low-entropy systems (credit cards, blockchain). In terms of high-entropy systems whose content is in a natural language setting such as chat systems or email systems. The application of the scheme on natural language was not explored at the time, as supported when Juels and Ristenpart (2014), stated that the application of the Honey Encryption Scheme on systems that employ human-generated content is an "interesting natural language processing problem". Since then, there have been numerous studies employing the scheme on systems with natural language content as the honey message. For instance, Omolara's (2018) study proposes an algorithm that produces decoys/fake messages for natural language messages in an email setting. Claiming to be the first to successfully generate a reasonable-length human-language decoy message that fools humans and automated tools.

Honey Passwords & Honeywords

In a typical system that employs HBAT, we have multiple possible passwords for each user account, with only one of which is the actual user password or sugarword. We define this set containing the sugarword and the sweet words as honey words. The attempted use of a sweetword to log in sets off an alarm, indicating to the system administrators that an adversarial attack has been reliably detected. In the context of the honey encryption scheme as proposed by Juels and Ristenpart (2014), the entry of an incorrect password by an adversary not only notifies the administrators, but also serves the adversary valid-looking but fake data termed as honey messages. Honey messages pertain to fake messages that are served to the attacker should a sweet word in a honey password be entered. This is so that the adversary cannot distinguish whether he has gained access to the actual user data or not. Honey messages are further discussed in a later section of this literature. Honey passwords may be used in such schemes to limit the production of honey messages only to decryptions using honey passwords.

The concept of honeywords stems from the cybersecurity concept of honeypots and more generally, the use of decoys to deceive the attacker. Juels and Rivest (2013) were the first to propose the approach of employing honeywords to deter dictionary attacks, or other similar brute force attacks even after database compromise. By determining that a breach had occurred should a login attempt use one of the sweet words or fake passwords. This approach was influenced by other similar existing techniques that employ deception. For instance, the honeypot technique which was popularized in the early 1990s demonstrated how an adversary can be led to attack the honeypot accounts and thus be detected (Cohen, 2006). Honeypot accounts are defined as fake accounts created by the system administrator for the purpose of detecting database breaches by an adversary. Then, the honeytoken and honeynet techniques, which are both types of honeypots and any interaction with either implies unauthorized or malicious activity (Spitzner, 2003). Most closely, Bojinov's (2010) Kamouflage serves as a theft-resistant password manager which utilizes decoy password lists alongside the correct password list to deceive the adversary as to which password list is genuine.

There exist numerous methods to generate honey passwords. All methods classify under one of the two categories: Legacy-UI and Modified-UI procedures (Juels & Rivest, 2013; Chakraborty & Mondal, 2017). In Legacy-UI, the user's password creation process is not influenced in any way by the system. The user does not know of the existence of the HBAT system nor the scheme. The system accepts only a username and password as user inputs. On the other hand, the Modified-UI, in addition to the conventional login credentials, requires the user to provide more information that will be used to generate the honeywords.

The purpose of honey generation methods is to produce a set of honey passwords for each user account. The manner in which they are generated can vary from one method to another. Some methods may derive the sweet words on the user's sugarword, as is the case for Juels and Rivest's (2013) chaffing methods, which replace certain positions in the sugarword with another character. And Chakraborty and Mondal's (2017) PDP model, which utilizes a modified-tail approach to add a password tail to the user's password. On the other hand, Erguler (2015) derives the sweet words from the passwords of other users and Guo et al (2019), which hides the relationship of users to passwords and uses honeypot accounts with fake usernames and passwords to identify malicious activity. There also exist other applications of honey password generation methods, such as those by Tian et al. (2021) for graphical passwords. However, the study shall focus on textual passwords only.

To be effective, honey passwords should be indistinguishable from each other. Thus, methods that function by deriving sweet words from the sugar word are common. Juels and Ristenpart (2014), state that although the production fake plaintext for every incorrect password is good for security, usability suffers when a legitimate user is served the plausible text through a user-typo, for instance. This typo-safety problem of the honey password generation methods is cited as one of the main challenges for every approach and is considered as a usability metric (Lindholm, 2019; Chakraborty & Mondal, 2017; Choi et al., 2017; Juels & Ristenpart, 2014).

Levenshtein Distance

The Levenshtein Distance is defined as the measure of similarity between two strings. The Levenshtein Distance between two strings is determined by the minimum number of single-character edits, these being the classic insertions, deletions or substitutions to transform one string to the other (Gilleland, n.d.). Typically used for analysis on the word-level rather than on the sentence-level or individual characters. It was employed in the three methods proposed by Akshima et al. (2018) (UPM, EPM, APM) as a way to address the typo-safety problem by ensuring that the minimum Levenshtein distance of >=2 is maintained between the password and honeywords generated by the three proposed methods. In the study of Omolara et al. (2020a) they propose an HE scheme that utilizes NLP tools to produce fictitious messages for a chatting application. The researchers also employed the Levenshtein distance to show that a large difference between the actual message and the decoy message is present in the word level. Attaining higher Levenshtein distance values compared to other HE schemes that also use NLP in generating honey messages.

That said, Clarkson (2005) in their study of error patterns found that in a mini-QWERTY keyboard the human error rates were as follows: 'Substitution' errors - 40.2%, 'Insertion' errors - 33.2%, 'Deletions errors - 21.4% and 'Transposition' errors - 5.2%. The Levenshtein Distance only accounts for Insertions, Deletions and Substitutions and not Transposition errors.

Flatness

Flatness refers to the expected probability that an adversary, considering their strategy and available information, will correctly identify the sugarword from a list of sweet words produced by a honeyword generation method (Erguler, 2015; Akshima et al., 2018; Chakraborty, 2015). It serves as an important evaluation metric for assessing the effectiveness of a honeyword generation method. Introduced in the study by Juels and Rivest (2013) as the first of three evaluation metrics, flatness quantifies a method's ability to detect intrusion attempts. Ideally, a lower flatness indicates a more secure system, as it makes it harder for adversaries to distinguish the sugarword from honeywords.

Juels and Rivest (2013) suggest that an optimal *k* value of 20 is ideal, resulting in a success probability of $\epsilon = 1/20$ or 5% of the adversary picking the sugarword. Increasing *k* would further reduce the adversary's chances, but this comes at the cost of higher system overhead. In a perfectly flat system, an adversary is ensured to have a minimum chance of $(k - 1)/k$ to pick a honeyword and get caught by the system.

Flatness, alongside DoS Resistance and Storage (Cost), the three are considered the most important characteristics of a honeyword method (Yasser et al., 2022). These metrics were introduced in Juels and Rivest's 2013 study and have since been used to evaluate the honeyword generation methods.

DoS Resistance

According to Akshima et al. (2018), a Denial-of-Service (DoS) attack seeks to make a service unavailable to legitimate users, which can be a concern for systems that implement honeyword generation techniques. In the context of the Honey Encryption Algorithm, when a security breach is detected, the system responds according to predefined security protocols, such as temporarily restricting user access to the system. An adversary could exploit this by deliberately entering a honeyword during authentication, triggering a DoS attack without having to fully compromise the system's database.

Storage Cost

Storage Cost represents a key aspect of system overhead that must be considered when evaluating the effectiveness of a honey password generation method. Introduced in Juels and Rivest (2013) alongside the two metrics previously discussed, storage is the third evaluation metric used to assess a honey password generation method. It evaluates the storage cost that a system incurs with the implementation of the honey password method.

In a standard password system, the storage cost is represented by "*hN*", where *h* denotes the hash length in bytes and *N* is the number of users in the system (Erguler, 2015). Since the storage requirements for usernames remain consistent across different methods, the studies focus solely on the storage cost of passwords per user, excluding the cost of usernames.

In a system that employs honey passwords, the storage cost is denoted by *khN*, with *k* representing the number of sweet words per user (Erguler, 2015; Juels & Rivest, 2013; Chakraborty & Mondal, 2015). Storage costs increase linearly by a factor of *k* in systems using honey passwords. This is because the system must store *k-1* additional passwords for each account. Chakraborty and Mondal (2015) mentions this great increase in storage cost as among the major drawbacks of honeyword generation approaches.

Comparative Analysis of Existing Methods

This section of the literature conducts a comparative analysis of the existing honey generation methods mentioned. The methods are evaluated based on two characteristics: Security and Usability. In the security analysis, the metrics of Flatness, DoS Resistance, Storage Overhead and Multiple-system Vulnerability are used. For the usability analysis, they are evaluated based on: Typo-safety, System-interference and their stress on memorability on users.

The table below provides a summary of existing studies and their proposed honey generation methods.

Table 1: Summary of Findings of Honeyword Generation Methods

Approach	**Author**	**Key Findings**
Tweaking Method	Juels and Rivest (2013)	This approach works by altering the real password through selective character tweaks to create honeywords. The user's actual password serves as the base for the generation algorithm. Characters are replaced according to their type: Character to Character, Numbers to Numbers and Special Characters to Special Characters.
Take-a-tail		This technique in which the system alters the ending (or "tail") of a user's actual password to generate honeywords. The tail could be a series of digits or characters appended to the original password. The user's actual password serves as the base for the generation algorithm

Modelling-Syntax		In this approach, a user's password is broken down into various tokens. Each token represents a specific syntactic element such as words, digits, or sequences of special characters. The system then generates decoys using similar types of tokens.
Tough-nuts		This method is designed to be computationally impossible to reverse from their hash values. For example, random bit strings of a fixed length are used as the hash values for these honeywords. The number and placement of these tough nuts are chosen at random. This introduces a special honeywords called "Tough Nuts"
Close-number-formation	Chakraborty and Mondal (2015)	The technique involves generating honeywords by making slight modifications to the numbers in a user's actual password to create new, similar-looking decoy passwords. By making small adjustments to the digits, the system produces plausible alternatives that are hard to differentiate from the genuine password. For instance, if a user's password features a numeric sequence such as "12345," the CNF method might produce honeywords like "12445" or "12346."
Caps-Key		This method creates a honeyword by altering the capitalization of characters in a user's actual password. This involves replacing specific characters in the original password with their uppercase or lowercase value to generate the honeywords.
Modified-tail		This technique is an enhancement of the Take-a-tail method. In this approach, the user must remember *m* characters from a set (S) containing *m + 1* characters in addition to their password. During login, the user appends these *m* characters to the password before submitting their credentials. The system then appends the remaining character from the set to the submitted information. Honeywords are generated by the system from all possible combinations of the *m + 1* characters.
Storage-index	Erguler (2016)	This method involves associating an index with each user, which points to the position of the real password within a list of stored passwords (which includes decoy honeywords). Instead of marking the real password directly, this index allows the system to verify the correct password without revealing which one it is.
PDP	Chakraborty and Mondal (2017)	For this approach the honeyword generation requires three pieces of information for the login process: the username, password, and a password-tail. This method allows the user to choose their password-tail, and by remembering just one tail, they can access multiple accounts. During registration, the user selects a password-tail of length l (>1) from a set of characters that includes letters (a-z) and digits (0-9). These characters are randomly arranged in a circular list called the Honey Circular List (HCL), which plays a role in generating honeywords. Based on the user's chosen password-tail, the system calculates a paired distance between its elements. From this paired distance, the system can generate |HCL| possible password-tails, starting from any character in the HCL. If an adversary compromises the password file, they will obtain the username, password, and the stored paired distance, but the paired distance will confuse them by suggesting 36 possible password-tails. This extra information (the password-tail) helps to mislead the attacker by presenting 36 potential honeywords.

Table 2: Comparison of Methods using Evaluation Metrics

Honeyword Method	Flatness	DoS Resiliency	Storage Overhead (Bytes)	Typo Safety	System Interference	Stress on memorability
Tweaking Method	0.05 if U = G	0.51	380	0.8	No	Low
Take-a-tail	0.05 (Unconditionally)	0.94	380	0.98	High	High
Modelling- Syntax	0.05 if U = G	=1	380	=1	No	Low
Tough-nuts	N/A	=1	20 × T^S	=1	No	N/A
Close-number-formation	0.05 if U = G	0.57	380	0.83	No	Low
Caps-Key	0.05 (Unconditionally)	0.002	380	0.29	Low	Low
Modified-tail	0.17 (Unconditionally)	0	100	0	Low	Low
Storage-index	0.05 if U = G	=1	84	=1	No	No
PDP	0.028	0.91 (*)	4	0.97	Low	Low

Flatness: Most of the methods exhibit a conditional flatness value of 0.05, indicating they approach an ideal level of flatness when the distribution of honeywords resembles that of genuine passwords. The PDP method, however, demonstrates a lower flatness value of 0.028, suggesting it offers a more diverse and less predictable distribution. In contrast, Caps-Key and Modified-tail exhibit distinct behaviors; Caps-Key maintains a flatness value of 0.05 unconditionally, whereas Modified-tail shows a higher flatness value of 0.17, indicating a less uniform distribution.

Denial-of-Service Resiliency: Regarding DoS Resiliency, most methods achieve values close to 1, indicating strong protection against Denial-of-Service (DoS) attacks. However, Caps-Key and Tough-nuts are less effective in this area, with Caps-Key showing very low resiliency (0.002), and Tough-nuts having a resilience level that depends on the parameter T^S.

Storage Overhead: In terms of Storage Overhead, there are considerable differences based on the number of bytes required for each method. While most methods need approximately 380 bytes, the storage-index approach exhibits a much lower overhead (1 byte). Notably, the PDP method significantly reduces storage requirements, needing only 4 bytes.

Typo Safety: When it comes to Typo Safety, methods like Modelling-syntax and PDP demonstrate high typo safety, which means they are better at handling mistyped passwords without raising false alarms. In contrast, Caps-Key has a lower typo safety score (0.29), making it more prone to issues related to typing mistakes.

System Interference: Regarding System Interference, the Take-a-tail method stands out for its high level of interference, requiring users to remember extra information, which could negatively impact the user experience. Conversely, other methods such as PDP, Tough-nuts, and storage-index do not exhibit system interference.

Stress on Memorability: For Stress on Memorability, the Take-a-tail method imposes significant stress on users due to system interference, making it more difficult to remember. Most other methods, however, cause low stress on memorability, making them more user-friendly

METHODOLOGY

To solve the issue of usability in the algorithm, we propose a new honeyword generation method which leverages a deep learning technique to generate the honeywords. Then, in ensuring that the typo-safety issue is addressed, we will be conducting word-level analysis of the honeywords utilizing the Damerau-Levenshtein Distance to account for four of the major typo causes: insertions, deletions, substitutions, and also transpositions. In doing so, the honey encryption's fictitious environment will only be served to adversaries attempting decryption with a honeyword.

In generating the honeywords, we employ a machine learning model that utilizes a type of Recurrent Neural Network known as LSTM or Long short-term memory to predict the next character of a sequence by identifying long-term dependencies based on the training data provided. The process for building the model, training and generation of honeywords are as follows:

Table 3: HPG Method Notation

Parameter	**Meaning**
tdl	Damerau-Levenshtein Threshold
k	Number of Honeywords
char_to_idx	Characters to Indexes
idx_to_char	Indexes to Characters
chars	Character Vocabulary
max_length_password_list	The length of the longest password in the password list.
input	data to be accepted by the model or layer
output	data to be outputted by the model or layer
embedding_dim	Size of the Embedding Dimension
lstm_units	Count of LSTM units
epochs	Number of complete passes required.
batch_size	Number of passwords per batch

We first initialize the parameters for our method, this includes the threshold for Damerau-Levenshtein, the number of honeywords to generate, the number of LSTM units, size of the embedding dimension.

Next, we prepare the dataset by constructing a vocabulary of all characters expected to appear in typical passwords. In this, the researchers employed the use of Python's string library methods: e reference ascii_letters (uppercase and lowercase letters), digits (0-9), and punctuation (special characters such as !, @, #, etc.), combining them into a single *chars* variable that defines our character set, and sorting them in ascending order according to their ASCII designation for consistency purposes. Notably, this vocabulary excludes whitespace and non-English characters. This is not a limitation, but rather a reflection of a policy decision in alignment with system password requirements. Should any additional characters be needed, they can easily be incorporated into the vocabulary set. Thus, meeting any specific policies or user needs concerning allowed characters in a password.

Table 4: Vocabulary Set

Variable	**Value**
digits	*0123456789*
ascii_letters	*lowercase + uppercase characters*
punctuation	!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

Afterwards, we generate a mapping of each character to an index, represented by a dictionary *char_to_idx*, and a mapping for each index to a character: *idx_to_char*. For the given *chars* set above, we generate the following mappings:

Table 5: Dictionary Mappings

Variable	**Value**
char_to_idx	{!: 0, ": 1, #: 2, $: 3, %: 4, &: 5, ': 6, ... z: 89, {: 90, |: 91, }: 92, ~: 93}
idx_to_char	{0: !, 1: ", 2: #, 3: $, 4: %, 5: &, 6: ', ... 89: z, 90: {, 91: |, 92: }, 93: ~}

Then, for each password in the given password list, we generate a sequence of indices according to the generated mapping. Suppose a password "password123", A sequence generated then would be: [79, 64, 82, 82, 86, 15, 81, 67, 16, 17, 18]. After all passwords have been converted to a sequence of indices, we proceed to create the input-output pairs that the model will consume to predict the next character in the sequence.

For each character of a password, we construct a subsequence. Given the same password "password123", a subsequence of input and output shall be:

Table 6: Subsequence Input and Output Mapping

Variable	**Characters**
input	*[[79], [79, 64], [79, 64, 82],* *[79, 64, 82, 82], [79, 64, 82, 82, 86],* *[79, 64, 82, 82, 86, 78],* *[79, 64, 82, 82, 86, 78, 81],* *[79, 64, 82, 82, 86, 78, 81, 67],* *[79, 64, 82, 82, 86, 78, 81, 67, 16], [79, 64, 82, 82, 86, 78, 81, 67, 16, 17]]*
output	*[64, 82, 82, 86, 78, 81, 67, 16, 17, 18]*

And then, it is correspondingly interpreted as:

Table 7: Subsequence Input and Output Mapping

Input	**Output**
79 → p	64 → a
79, 64 → pa	82 → s
79, 64, 82 → pas	82 → s
…	…
79, 64, 82, 82, 86, 78, 81, 67 → password	16 → 1
79, 64, 82, 82, 86, 78, 81, 67, 16 → password1	17→ 2
79, 64, 82, 82, 86, 78, 81,67, 16, 17 → password12	18→ 3

Afterwards, we pad the input sequences as the model expects inputs of consistent length. To achieve this, we first set *max_length_password_list* to the length of the longest password in the password list. Then we utilize tensorflow's *pad_sequence* method to pad the sequences with zeroes, with *max_length_password_list* as our desired length. Understand that the user password cannot be longer than the longest password of a given password list, else the honeywords will not be generated properly.

For a given *max_length_password_list* of 11, *input* then becomes a 2D NumPy array of integers that represents the padded subsequences of the password character indices:

Table 8: Padded NumPy Array Input

Variable	**Value**
input	*[[79 0 0 0 0 0 0 0 0 0 0]* *[79 64 0 0 0 0 0 0 0 0 0]* *[79 64 82 0 0 0 0 0 0 0 0]* *[79 64 82 82 0 0 0 0 0 0 0]* *[79 64 82 82 86 0 0 0 0 0 0]* *[79 64 82 82 86 78 0 0 0 0 0]* *[79 64 82 82 86 78 81 0 0 0 0]* *[79 64 82 82 86 78 81 67 0 0 0]* *[79 64 82 82 86 78 81 67 16 0 0]* *[79 64 82 82 86 78 81 67 16 17 0]]*

We also convert our *output* into a binary matrix representation of the input labels as a NumPy array whose number of classes or categories is equal to the length of *char_to_idx*. Each value in *output* is then converted into one-hot encoded vectors in matrix representation:

Table 9: One-Hot Encoded Vectors Matrix Representation of Output

Variable	**Value**
output	*[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.* *0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.* *0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.* *0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]* *[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.* *0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.* *0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.* *0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]* ... *]*

Here, our *output*'s first element '64', which is character 'a' in our mapping, is represented as the first element in one-hot encoded vector binary representation. The representation of each character's position in the matrix corresponds to their index in the vocabulary. These processes enable the model to work with numerical representations of our inputs and expected outputs.

The next step is constructing our model. Tensorflow and its libraries were employed to build our model. Summarily, our model consists of an embedding layer followed by two LSTM layers, and a dense layer.

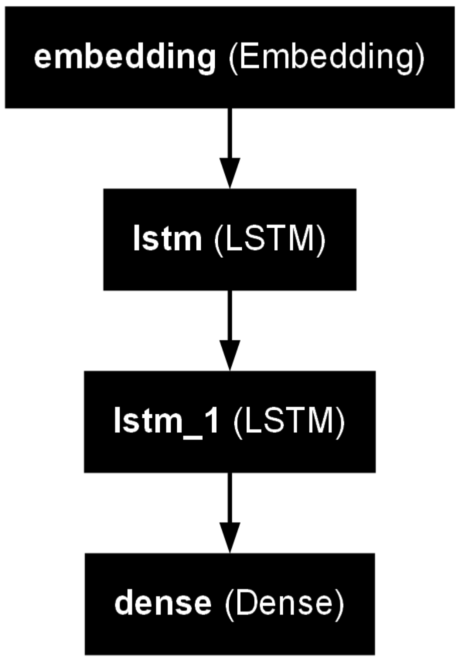


Figure 1: Model Architecture

The following table shows the input and output of each layer of the model:

Model Layer	Input	Output
Embedding	2D tensor with shape: (batch_size, input_length).	3D tensor with shape: (batch_size, input_length, output_dim).
LSTM	3D tensor with shape: (batch_size, input_length, output_dim).	Sequence of hidden states for each character in the input sequence. Shape: (batch_size, max_length, lstm_units)
LSTM_1	The sequence of hidden states from the first LSTM layer. Shape: (batch_size, max_length, lstm_units)	A single hidden state vector representing the entire sequence. Shape: (batch_size, lstm_units).
Dense	The single hidden state vector outputted by the second LSTM layer. Shape: (batch_size, lstm_units).	With the softmax activation function, it outputs a vector of probabilities, one for each character in the vocabulary. Shape: (batch_size, vocabulary_size)

The model is then compiled with the general optimizer function Adam, which adaptively adjusts the learning rates, and Categorical Crossentropy for the loss function to minimize classification error. We also employ an early stoppage function in the case that the model's loss value does not improve over a specified 5 epochs. This ensures no unnecessary training is conducted, and only the best performing model is used.

The model takes a sequence of characters, embeds them into dense vectors, processes them through LSTM layers to capture sequential patterns, and finally outputs a probability distribution for the next character. By leveraging both embeddings and temporal dependencies captured during training, the model is able to generate realistic honeywords based on actual passwords used by humans.

The model may then be trained on a dataset of pre-processed password lists such as those from mySpace, RockYou, phpBB. However, as mentioned by Juels and Rivest (2013), and Dionysiou et al. (2021), generating honeywords based on a published list is not a good idea as it may be that the adversary may also have access to these lists and use them to identify the honeywords. For this study, the model will be trained on the cleaned phpBB passwords sourced from SecLists. However, actual implementations of the model should have the training dataset sourced from the password database for reasons of risk mitigation, and to have the model abide by specific password policies (Dionysiou et al., 2021).

A honeyword is generated by the model then by providing it with a seed, typically the first few characters of the sugarword. We have adopted a seed of length 3. To allow the model to generate the rest of the characters on its own. The length of the sugarword is retained. Minus the seed text, the remaining characters of the honeyword are determined by sampling the next character's index through a probability distribution generated by the model using temperature scaling. Temperature scaling adds randomness to the honeyword generation, a temperature of 0.7 is recommended as temperatures less than one (<1) sharpens the probability distribution therefore favoring predictions of higher probability, while greater than one (>1) temperature flattens the distribution and increases randomness. This is repeated until the sugarword's length is reached, at which point a honeyword candidate is generated.

After a honeyword candidate is generated, it is assessed for its Damerau-Levenshtein distance. The Damerau-Levenshtein distance is calculated by:

$$d_{a,b}(i,j) = \min \begin{cases} 0 & \text{if } i = j = 0, \\ d_{a,b}(i-1,j)+1 & \text{if } i > 0, \\ d_{a,b}(i,j-1)+1 & \text{if } j > 0, \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} & \text{if } i,j > 0, \\ d_{a,b}(i-2,j-2)+1_{(a_i \neq b_j)} & \text{if } i,j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j, \end{cases}$$

Figure 2: Damerau-Levenshtein Distance Formula

Let d(i, j) be the distance between the first i characters of string A and the first j characters of string B. The first row corresponds to the initial state, where distance is zero if two strings require zero edits to be equal. Afterwards, each recursive call then matches one of the cases covered by the Damerau–Levenshtein distance: the second row covers deletion of string A and string B, the third; insertion, and the fourth to a match or a mismatch on whether the characters correspond to a transposition.

We adopt the recommended value of >= 3 for the Levenshtein Distance of Akshima et al. (2018), for our Damerau-Levenshtein distance value. If the Damerau-Levenshtein distance of a given sweetword and the user password is >=3 then the sweetword is added to the set, otherwise another sweet word is generated. It is possible for a sweetword to retain most of its sugarword characters. Thus, this ensures that the minimum safe distance between the user password and the honeywords is maintained, while covering all four of the major typo-cases of Insertion, Deletion, Substitution and Adjacent Transposition.

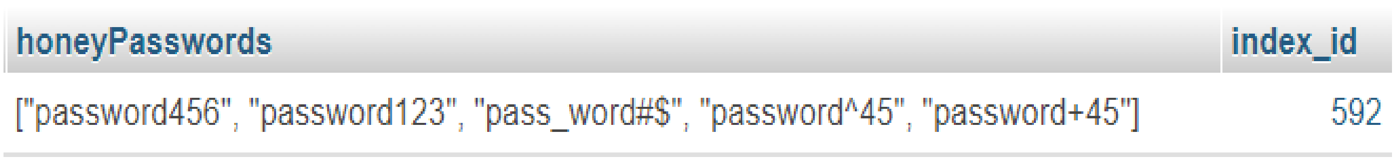
Once the word is acceptable - the honeyword candidate's and sugarword's Damerau-Levenshtein distance is equal to or above the threshold - the honeyword is added to the set. Upon generating k-1 honeywords, the sugarword is added to the list which is shuffled to randomize the positions. For authentication, we save the position of the sugarword through the sugar index.

The honeywords are then assigned to the user's account, with the sugar index and the user's random index being stored in the honey checker - an auxiliary server that is commonly used in HE schemes to check whether the inputted password is the actual password.

For a given password "password123", a possible tuple to be generated may be:

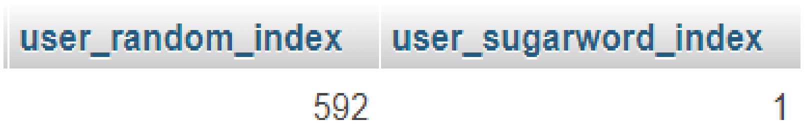
(['password456', 'password123', 'pass_word#$', 'password^45', 'password+45'], sugarindex: 1)

Where the list of honeywords is saved to a honeypasswords table. The sugarword index and user random index are saved to the honeychecker table.



honeyPasswords	index_id
["password456", "password123", "pass_word#$", "password^45", "password+45"]	592

Figure 3: Honeypasswords SQL Table entry



user_random_index	user_sugarword_index
592	1

Figure 4: Honeychecker SQL Table entry

To determine whether the objective of addressing the usability issue has been sufficiently achieved, the following section shall evaluate the method using the same evaluation metrics discussed in this paper.

RESULTS AND DISCUSSION

In this chapter, results from the problems and the proposed objective is discussed by the proponents. The findings in this study are interpreted and described as new insights emerged from the results of the study's problems.

To evaluate whether the first objective of improved usability has been achieved, we employ a modified data collection approach based on Chatterjee et al. (2016), supplemented by Fahl et al. (2013). Qualtrics has been utilized in creating the survey. We require that the respondents use a standard QWERTY keyboard layout and be on desktops. In the first part of the survey, the respondents were asked for general demographic information such as their age, name etc. Then, specific information depends on whether they were students or employees.

Then, the respondents were asked to enter a total of 20 different passwords under a given time limit. Password masking is then employed to mimic actual authentication settings; input characters will be masked into asterisks, with no option to reveal the actual password characters.

Instead of disseminating the survey through online crowdsourcing marketplaces such as Amazon MTurk, the researchers opted to conduct the survey using random sampling through Qualtrics. We disseminated our survey to Pamantasan ng Lungsod ng Maynila students and office employees from an anonymous brokerage firm. In total we had 100 respondents. After processing, 30 responses were found to be invalid because of either incomplete submission, failure to follow instructions of the survey, or blatant disregard for answering correctly or are overly uniform answers and were not factored into the results.

The researchers did not opt for a data collection approach where the users are asked to input their accounts' passwords multiple times. This is as it would complicate and slow the data collection process significantly while not ensuring real ecological validity; guarantees of whether test results are indicative of that of real-life settings (Chatterjee et al., 2016).

The passwords selected to be inputted into the survey will be sourced from the cleaned phpBB data set sourced from SecLists. Further, they shall mimic general password guidelines: The passwords will be 9 to 15 characters in length; Be composed of any combination of digits, lower and uppercase characters, and symbols. Reference the vocabulary set in the previous section for a complete list. Passwords will not contain spaces, or any character that is not in the vocabulary set.

In total, the researchers will compare 3 legacy-UI methods and one Modified UI method. The two legacy-UI methods and one modified-UI method from Juels and Rivest (2013): Chaffing-by-Tail-Tweaking, Chaffing-with-a-Password-Model, and Take-A-Tail respectively, and the other legacy-UI is the proposed method. Due to the presence of a modified-UI method, the data results will not be accurate if all honeyword methods will be employed for each password. As the Take-A-Tail method requires a tail to be appended, whereas the other three do not.

The 20 passwords' honeywords will be generated as follows:

1. 5 passwords will employ Chaffing-by-Tail-Tweaking;
2. 5 passwords will employ Take-A-Tail Method;
3. 5 passwords will employ Chaffing-with-a-Password-Model;
4. 5 passwords will use the proposed method.

For passwords in Take-a-Tail, they are taken in the same manner as the passwords of the other methods, then appended by randomly generated digits at the end as required by the method. The table below shows the base passwords used per method with table # showing the honeywords for each base word.

Table 10: Method and Base Words

Method	Base Words
Chaffing-by-tail-tweaking	altinordu52 HPw2207!@# MIlkSHake1 Makkadeh!@# jeff@farm1
Take-a-Tail	riaflashmx846 1441262626027 lamurg0d694

	#ector@1258 swordhunter583
Chaffing-with-a-Password-Model	hazo996655 th3matr1x papa072112 dovesciare9090 EAfiGI5389
ML Method	DraiksRock251 diabolo666 letmeloveyou password8169 rhfm281010

The graph below shows the number of occurrences for the four major typo-classes: Insertions, Deletions, Substitutions and Transpositions that were observed.

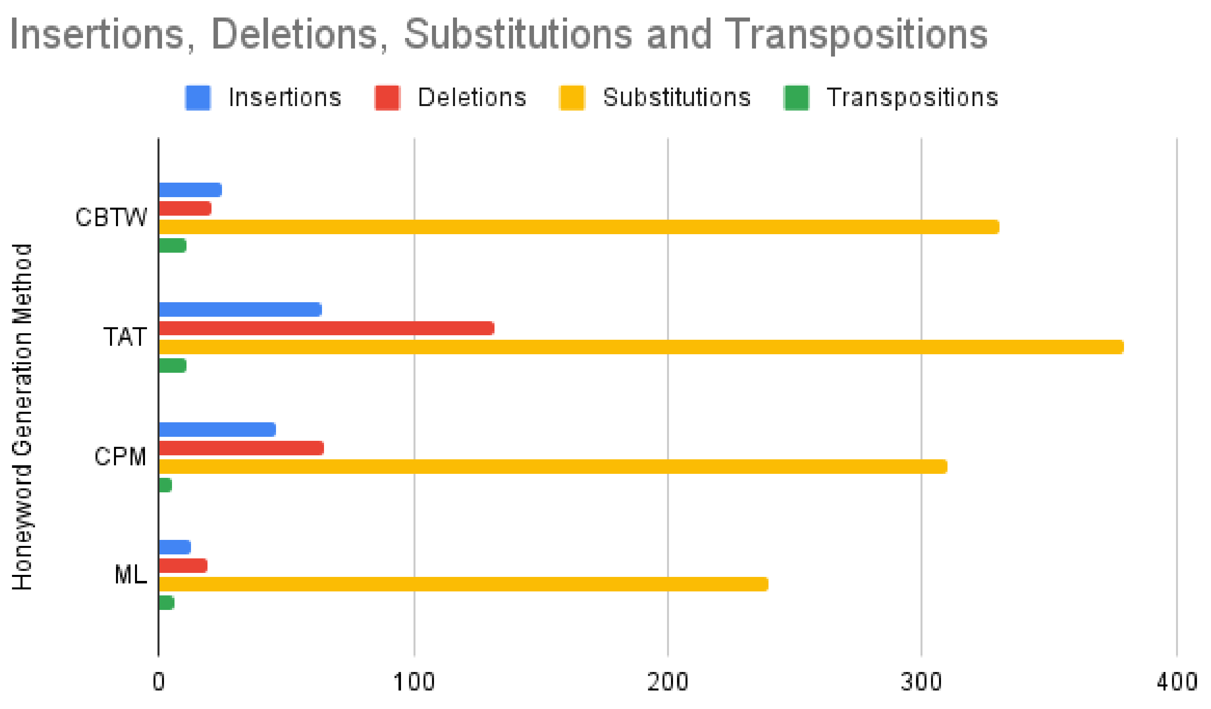


Figure 5: Graph of Insertions, Deletions, Substitutions, and Transpositions in User Inputs.

In the graph above which shows the number of Insertions, Deletions, Substitutions and Transpositions that were committed by 70 respondents across a sample of 20 passwords, the y-axis represents the honeyword generation techniques (CBTW, TAT, CPM, ML), while the x-axis represents the total number of operations and their corresponding percentages. These errors are an essential indication of each honeyword generation method's security, usability and predictability. A higher number of errors indicates greater predictability, which compromises the strength of the generated decoys and, thus, system security.

Among the total 1677 errors observed in insertions, deletions, substitutions and transpositions, TAT was found to be the weakest method out of the errors that were observed, with the highest number of errors making up 34.93% of the total. This shows that TAT creates decoys that are the most predictable and susceptible to attacks, especially when users type incorrect passwords while trying to choose the correct one. Following contributions of 23.08% and 25.40%, respectively, CBTW and CPM continued to show difficulties when generating effective honeywords. However, with only 16.58% of the total errors, our suggested machine learning method performed significantly better than the others. This reduces the likelihood that the user will easily identify the correct password from the list of honeywords, making it the most effective method for improving the quality of the decoys and decreasing predictability, particularly when users make typos.

These results also show the challenges of honeyword generation in addressing two key limitations according to Akif et al. (2019). Co-relational Hazard and Distinguishable well-known password pattern. When the username and the password are strongly associated, a co-relational hazard occurs. For instance, the actual password becomes obvious among the decoys if TAT generated honeywords are unable to break these associations. The real password is more predictable because of TAT's high error rate (34.93%), which indicates that it is unable to successfully break these associations. Conversely, distinguishable well-known password patterns arise when a user's password has a common format, such as "NameYear" or "123Password." Methods such as CPM and CBTW, which have error contributions of 25.40% and 23.08%, respectively, find it difficult to produce honeywords that sufficiently differ from these patterns to reveal the actual password.

Therefore, when compared to the other approaches, the ML approach's advantage is evident. ML decreased mistakes by 28.15% compared to CBTW, 34.74% compared to CPM, and 52.54% compared to TAT. These notable decreases show how machine learning (ML) may provide honeywords that are much more unpredictable, making it more difficult for users to figure out the right password—especially when they type it incorrectly—thus improving system security and adhering to the fundamental idea of honey encryption.

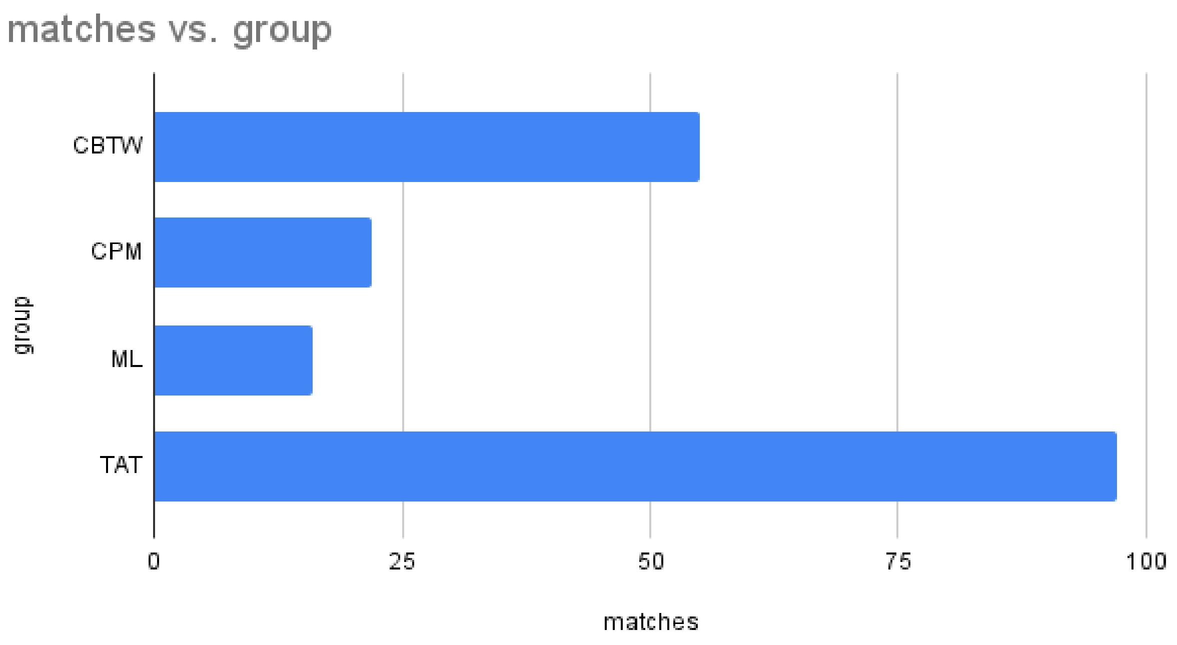


Figure 6: Graph of total matches for each method group (CBTW, TAT, CPM, and ML)

The graph above illustrates how effective different methods are in comparison, with fewer matches signifying higher security and stronger decoy production. The y-axis shows the groups corresponding to various honeyword generation techniques (CBTW, CPM, ML, and TAT), while the x-axis shows the number of matches (instances where a user's typo matches a generated honeyword for the base word/password) which shows the relationship between honeyword generation techniques and their susceptibility to user typos.

In our sample size of 100 honeywords, each group consisted of 5 base words, with 5 honeywords per base word. With 97 matches (51.05% of the total), the results indicate that user typos commonly matched honeywords in the TAT group. This suggests that TAT generates highly predictable honeywords that are ineffective at imitating real passwords. Similarly, with 55 matches (28.95%), CBTW showed notable weaknesses. On the other hand, CPM showed improved performance, with only 22 matches (11.58%), while our proposed ML method demonstrated the best performance, with just 16 matches (8.42%). This illustrates how the proposed machine learning can produce honeywords that are less likely to match user typos.

These findings show that weaker honeyword generating methods like TAT and CBTW generate honeywords that are more likely to be matched when user input contains base word typos. This makes it more likely that someone will figure out the correct password, which lowers the system's security. The ML approach, on the contrary hand, creates honeywords that successfully obscure the base word, leading to fewer matches and improved security.

Therefore, our proposed machine learning approach reduces matches by 27% when compared to CPM, 71% when compared to CBTW, and 83% when compared to TAT. ML firmly adheres to the principles of Honey Encryption by limiting the likelihood that user typos would match honeywords. This ensures that decryption attempts appear equally plausible and significantly lowers the likelihood of successful attacks. Thus, our method has a lower probability of legitimate users' typos resulting in a honeyword, resulting in a significant improvement in usability in password-based authentication systems using the enhanced Honey Encryption Algorithm.

CONCLUSION

In summary, this study presents a way to address the usability of issues present in the Honey Encryption Algorithm through the use of our enhanced honeyword generation method. By leveraging machine learning techniques and the Damerau-Levenshtein distance metric in our enhanced algorithm, we achieve a typo rate of 29%, with only 4.57% of typos producing a honeyword candidate. In comparison with the original Honey Encryption Algorithm using Take-a-Tail method, which had a 71% typo rate, and with 27.71% of typos resulting in honeyword candidates. Thus, our method has a lower probability of legitimate users' typos resulting in a honeyword, resulting in a significant improvement in usability in password-based authentication systems using the enhanced Honey Encryption Algorithm.

CONFLICT OF INTEREST

The authors declare no conflicts of interest.

REFERENCES

1. Abiodun, E. O., Jantan, A., Abiodun, O. I., & Arshad, H. (2020a). Reinforcing the security of instant messaging systems using an enhanced honey encryption scheme: the case of WhatsApp. Wireless Personal Communications, 112, 2533-2556.
2. Akshima, Chang, D., Goel, A., Mishra, S., & Sanadhya, S. K. (2018). Generation of secure and reliable honeywords, preventing false detection. IEEE Transactions on Dependable and Secure Computing, 16(5), 757-769.
3. Akif, O. Z., Sabeeh, A. F., Rodgers, G. J., & Al-Raweshidy, H. S. (2019). Achieving flatness: Honeywords generation method for passwords based on user behaviours.
4. Bojinov, H., Bursztein, E., Boyen, X., & Boneh, D. (2010). Kamouflage: Loss-resistant password management. In Computer Security–ESORICS 2010: 15th European Symposium on Research in

Computer Security, Athens, Greece, September 20-22, 2010. Proceedings 15 (pp. 286-302). Springer Berlin Heidelberg.

5. Chakraborty, N., & Mondal, S. (2017). On designing a modified-UI based honeyword generation approach for overcoming the existing limitations. Computers & Security, 66, 155-168.
6. Chakraborty, N., & Mondal, S. (2015, September). Few notes towards making honeyword system more secure and usable. In Proceedings of the 8th International Conference on Security of Information and Networks (pp. 237-245).
7. Chatterjee, R., Athayle, A., Akhawe, D., Juels, A., & Ristenpart, T. (2016, May). pASSWORD tYPOS and how to correct them securely. In 2016 IEEE Symposium on Security and Privacy (SP) (pp. 799-818). IEEE.
8. Choi, H., Nam, H., & Hur, J. (2017, January). Password typos resilience in honey encryption. In 2017 International conference on information networking (ICOIN) (pp. 593-598). IEEE.
9. Clarkson, E., Clawson, J., Lyons, K., & Starner, T. (2005, April). An empirical study of typing rates on mini-QWERTY keyboards. In CHI'05 extended abstracts on Human factors in computing systems (pp. 1288-1291).
10. Cohen, F. (2006). The Use of Deception Techniques: Honeypots and Decoys. Handbook of Information Security, 3(1), 646-655.
11. Dionysiou, A., Vassiliades, V., & Athanasopoulos, E. (2021, May). Honeygen: Generating honeywords using representation learning. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (pp. 265-279).
12. Erguler, I. (2015). Achieving flatness: Selecting the honeywords from existing user passwords. IEEE Transactions on Dependable and Secure Computing, 13(2), 284-295.
13. Fahl, S., Harbach, M., Acar, Y., & Smith, M. (2013, July). On the ecological validity of a password study. In Proceedings of the Ninth Symposium on Usable Privacy and Security (pp. 1-13).
14. Garrett, P., Seidman, J. (2011). EMR vs EHR – What is the Difference? Retrieved April 08, 2024, from https://www.healthit.gov/buzz-blog/electronic-health-and-medical-records/emr-vs-ehr-difference
15. Guo, Y., Zhang, Z., & Guo, Y. (2021). Superword: A honeyword system for achieving higher security goals. Computers & Security, 103, 101689.
16. HealthIT.gov. (2017 September 17). What is a patient portal? Retrieved April 10, 2024, from https://www.healthit.gov/faq/what-patient-portal#:~:text=A%20patient%20portal%20is%20a,anywhere%20with%20an%20Internet%20connection.
17. Juels, A., & Ristenpart, T. (2014). Honey encryption: Security beyond the brute-force bound. In Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33 (pp. 293-310). Springer Berlin Heidelberg.
18. Juels, A., & Rivest, R. L. (2013, November). Honeywords: Making password-cracking detectable. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 145-160).
19. Lindholm, R. (2019). Honey Encryption: implementation challenges and solutions (Master's thesis).
20. McCallister, E. (2010). Guide to protecting the confidentiality of personally identifiable information. Diane Publishing.
21. MedlinePlus Medical Encyclopedia. (2022). Patient portals - an online tool for your health. Retrieved April 09, 2024, from https://medlineplus.gov/ency/patientinstructions/000880.htm
22. Omolara, A. E., Jantan, A., & Abiodun, O. I. (2019a). A comprehensive review of honey encryption scheme. Indonesian Journal of Electrical Engineering and Computer Science, 13(2), 649-656.
23. Omolara, A. E., & Jantan, A. (2019b). Modified honey encryption scheme for encoding natural language message. International Journal of Electrical and Computer Engineering (IJECE), 9(3), 1871-1878.

24. Omolara, A.E., Jantan, A., Abiodun, O. I., Arshad, H., Dada, K. V., & Emmanuel, E. (2020b). HoneyDetails: A prototype for ensuring patient's information privacy and thwarting electronic health record threats based on decoys. Health informatics journal, 26(3), 2083-2104.
25. Omolara, A. E., Jantan, A., Abiodun, O. I., & Poston, H. E. (2018, March). A novel approach for the adaptation of honey encryption to support natural language message. In Proceedings of the International multiconference of engineers and computer scientists (Vol. 1, pp. 14-16).
26. Spitzner, L. (2003, December). Honeypots: Catching the insider threat. In 19th Annual Computer Security Applications Conference, 2003. Proceedings. (pp. 170-179). IEEE.
27. UC Berkeley. (n.d.). HIPAA PHI: Definition of PHI and List of 18 Identifiers. Retrieved April 10, 2024, from https://cphs.berkeley.edu/hipaa/hipaa18.html
28. Verizon. (2017). 2017 Data Breach Investigations Report. https://www.knowbe4.com/hubfs/rp_DBIR_2017_Report_execsummary_en_xg.pdf
29. Yasser, Y. A., Sadiq, A. T., & AlHamdani, W. (2022). Generating Honeyword Based on A Proposed Bees Algorithm. IRAQI JOURNAL OF COMPUTERS, COMMUNICATIONS, CONTROL AND SYSTEMS ENGINEERING, 22(4).