Ingénierie des Systèmes d'Information
Vol. 31, No. 1, January, 2026, pp. 137-145
Journal homepage: http://iieta.org/journals/isi

A Pelican Algorithm-Based Method for Secure Honeyword Generation

Raad Sadi Aziz[1], Wafaa M. Salih Abedi[2*], Sura Mazin Ali[3], Mohamad Ab. Saleh[4], Ahmed T. Sadiq[5]

[1] Technical Institute of Al-Suwaira, the Middle Technical University (MTU), Baghdad 10001, Iraq
[2] Department of Artificial Intelligence, College of Technology, City University Ajman, Ajman 00000, UAE
[3] Political Science College, Mustansiriyah University, Baghdad 10001, Iraq
[4] Banking and Financial Department, College of Economic and Administration, Al Iraqia University, Baghdad 10001, Iraq
[5] Department of Computer Science, University of Technology-Iraq, Baghdad 10001, Iraq

Corresponding Author Email: wafaa.m.salih@gmail.com

https://doi.org/10.18280/isi.310113

Received: 20 September 2025
Revised: 20 November 2025
Accepted: 18 January 2026
Available online: 31 January 2026

Keywords:
honeyword generation, password security, Pelican Algorithm, metaheuristic optimization, authentication security, password cracking detection, cyber security

ABSTRACT

Passwords remain the primary mechanism for protecting user accounts and sensitive digital resources. However, password databases are frequently targeted by attackers using offline cracking techniques. Honeywords have been proposed as an effective defense mechanism that introduces decoy passwords to detect unauthorized login attempts. Despite their advantages, many existing honeyword generation methods suffer from predictable structures, low diversity, and insufficient flatness, which may allow attackers to distinguish the real password from decoys. This study proposes a secure honeyword generation method based on the Pelican Algorithm (PA), a swarm-based metaheuristic optimization technique. The proposed approach decomposes a password into tokens consisting of alphabetic characters, digits, and special symbols, and generates honeywords using a hybrid strategy that combines PA-based token optimization with randomized digit and symbol generation. The algorithm is designed to improve key honeyword properties, including similarity to the original password, semantic acceptability, diversity, and stability. Experimental results demonstrate that the proposed method significantly improves the flatness property, which is the most critical factor in honeyword security. While the original Juels–Rivest honeyword scheme achieved a flatness of approximately 5% (1 in 20), the proposed approach reduces the probability of correctly identifying the real password to about 2% (1 in 49). Furthermore, even when partial token information is known, the probability of guessing the correct password remains limited to approximately 14%. These results confirm that the proposed PA–based framework provides a robust and scalable solution for enhancing password security against statistical guessing and database compromise attacks.

1. INTRODUCTION

Password authentication is the most widely used method due to its simplicity and ease of memorization; however, it has been extensively studied and targeted by various attacks, including password hacking techniques [1, 2]. To strengthen password security, honeyword-based authentication schemes have been proposed, which generate fake passwords (honeywords) alongside the real password. When an attacker compromises a password database, the presence of honeywords prevents them from easily identifying the real password. If the attacker attempts to log in using a honeyword, the system triggers a silent alarm, alerting the administrator without notifying the attacker [3-5]. A honeychecker server is typically used to securely distinguish real passwords from honeywords and to verify login attempts without exposing the real password to the main authentication server [6-8].

Despite their effectiveness, existing honeyword generation methods suffer from several limitations. Many approaches produce honeywords with low flatness, predictable patterns, and low semantic acceptability, making them vulnerable to statistical guessing attacks [9-11]. Furthermore, some methods lack diversity and adaptability, leading to repetitive or unrealistic honeywords that can be distinguished from real passwords. These weaknesses diminish the practical security benefits of honeyword-based systems, particularly when attackers employ advanced analysis and guessing strategies [12, 13].

To address these challenges, this study proposes an efficient honeyword generation approach based on the Pelican Algorithm (PA). The PA is a metaheuristic optimization technique known for its strong exploration-exploitation balance, adaptability, and fast convergence, making it suitable for generating high-quality honeywords. In the proposed method, passwords are decomposed into tokens (characters, digits, and symbols), and each token type is processed using tailored strategies. The PA is combined with random digit generation to produce honeywords that are semantically acceptable, diverse, and closely similar to the original password.

The main contributions of this paper are summarized as follows:

1. A novel honeyword generation method based on the PA is proposed.

2. The method addresses key limitations of existing techniques, including low flatness, low semantic acceptability, and weak diversity.

3. A token-based generation process is introduced, which handles characters, digits, and symbols separately to produce realistic honeywords.

4. An evaluation mechanism is developed to assess the quality of generated honeywords, including the integration of random digit addition.

5. Experimental results demonstrate significant improvements in flatness and guessing resistance compared with existing methods.

This paper outlines various methods for creating honeywords and explains the honeyword technique. It will detail the proposed system using PA, present experimental results, and compare them with previous honeyword generation methods before concluding.

2. RELATED WORKS

Numerous studies in recent years have proposed various methods for generating honeywords, and several asymptotic research efforts have been conducted in this field.

Erguler [14] proposed several methods for creating honeywords, including modifying passwords, using a dictionary, adding extra characters, utilizing system-generated honeywords, allowing user-generated honeywords, and combining these approaches. Based on user interface impact, they categorized these methods into two groups: Legacy-UI, which includes ChaIng-by-tail-tweaking and Hybrid generation methods, and Modified UI, featuring techniques like Take-a-tail and Random Pick.

Chakraborty and Mondal [15] introduced the Paired Distance Protocol (PDP), facilitating honeyword creation with an updated user interface. Users log in by providing a password and a username, which must consist of more than one character (t > 1) chosen from two digits (0-9) and letters (a-z).

In the study by Chang et al. [16], a " storage-index " method was proposed for generating honeywords using the actual passwords of current users, creating realistic honeywords that mimic existing system passwords.

In the study by Akif et al. [17], a method for creating honeywords includes the "evolving-password model," "user-profile model," and "append-secret model". The model of evolving passwords analyzes the frequency of password patterns to generate honeywords. The user-profile model uses various types of user data to create unique token sets. In the append-secret model, the user provides their username, password, and an additional secret to generate a random string, with H(password | result) stored in the password file.

Catuogno et al. [18] proposed using four approaches to create honeywords. The first set derives from user information, utilizing public personal questions to create a character and number database. The second set replicates a dictionary attack by altering the original password with three digits or characters. The third set consists of random honeywords from 500 common weak passwords, while the fourth set rearranges characters from the user ID to generate new honeywords.

Yasser et al. [19] proposed a honeyword generation method based on the music-inspired Harmony Search Algorithm (HSA), which improves the generation process and its properties. However, if the initial population lacks diversity, the method may require repeated executions, leading to wasted time and storage. Similarly, Brindtha et al. [20] introduced a swarm-based metaheuristic approach using the Meerkat Clan Algorithm to enhance the honeyword generation process and improve honeyword characteristics.

3. HONEYWORDS

The honeywords system generates fake passwords, or honeywords, from a real password known as the sugar word. These are stored in the user account as sweet words and hashed for security [21, 22]. If an attacker retrieves 42 plain passwords from hashed ones, they must identify the real password among the sweet words. A subtle alert may then notify the system administrator of a potential compromise [23, 24]. Administrators may block accounts, postpone access, or send notifications based on organizational policy [25, 26].

4. THE PELICAN ALGORITHM

This part outlines the motivation and mathematical framework of the PA algorithm, which employs a group of "pelicans" representing potential solutions to an optimization problem. As Eq. (1) indicates, each pelican begins with a random position within the defined limits.

$$x_{ij} = l_j + rand \bullet u_j - l_j;\ i = 1, \ldots, N;\ j = 1, \ldots, m \qquad (1)$$

where, x_{ij} represents the jth variable value defined by the ith candidate solution, m signifies the count of problem variables, N denotes the total number of members in the population, l_j is the lower limit for the jth variable, and u_j is the upper limit for the jth variable in the problem, a rand is a random number ranging from [0, 1].

The pelican population in the proposed protected area is identified using a matrix with rows for potential solutions and columns for problem variable values, Eq. (2).

$$X_{N\times m} = \begin{matrix} X_{1,1} & \cdots & X_{1,j} & \cdots & X_{1,m} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ X_{i,1} & \cdots & X_{i,j} & \cdots & X_{i,m} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ X_{N,1} & \cdots & X_{N,j} & \cdots & X_{N,m} \end{matrix}_{N\times m} \qquad (2)$$

X represents the population matrix of the pelican species, while Xi refers to the ith individual pelican. The objective function results are derived by means of the objective function vector in Eq. (3).

$$F = \begin{matrix} F1 \\ \vdots \\ Fi \\ \vdots \\ FN \end{matrix}_{N\times 1} = \begin{matrix} F(X1) \\ \vdots \\ F(Xi) \\ \vdots \\ F(XN) \end{matrix}_{N\times 1} \qquad (3)$$

The vector objective function is denoted by F, and F_i represents the objective function value of the ith candidate solution. These concepts, together with the pelican's target-approaching strategy, are expressed in the corresponding Eq. (4).

$$X_{ij}^{Pi} = \begin{cases} Xi + rand.(Pi - I.Xi,j), & ifFp < Fi \\ Xi,j + rand.(Xij - Pi), & else \end{cases} \quad (4)$$

where, $X_{ij}^{P_1}$ represents the updated state of pelican I in the dimension j during phase 1. I is a random variable that can be set to one or two, p_j denotes the target position in dimension j, and Fp indicates the value of the objective function. Effective updating helps the algorithm avoid moving into suboptimal regions by employing the updating type. This procedure is described using Eq. (5).

$$Xi = \begin{cases} X_i^{P1} \\ Xi, \quad else \end{cases}, \quad F_i^{Pi} < F_i \quad (5)$$

$X_i^{P_1}$ is the ith pelican's updated status, while $F_i^{P_1}$ Denotes the value of its objective function. The algorithm explores the nearby points around the pelican's position to find an improved solution. This hunting actions of pelicans is represented in Eq. (6).

$$X_{ij}^{P2} = X_{ij} + R.\left(1 - \frac{t}{T}\right).(2.rand - 1).X_{ij} \quad (6)$$

The term $X_{ij}^{P_2}$, refers to the updated position of the pelican (i) in dimension (j) during Phase 2. R is a constant with a value of 0.2, and the neighborhood radius around xi,j is defined by R·(1 - t/T), where T is the total allowed number of iterations, and t is the current iteration count. The term "R·(1 - t/T)" indicates the radius of neighborhood population members, which is used for local searching near each member to achieve improved solutions. As the algorithm progresses, the coefficient "R· (1 - t/T)" decreases, resulting in smaller neighborhood radii for each member.

In this phase, an efficient update mechanism is applied to accept or reject the pelican's new position, as shown in Eq. (7).

$$X_i = \begin{cases} X_i^{P2} \\ Xi, \quad else \end{cases}, F_i^{P2} < F_i \quad (7)$$

where, X_i^{P2} is the pelican I in the new status e, and F_i^{P2} is the value of the objective function.

5. THE PROPOSED HONEYWORD GENERATION SYSTEM

The proposed honeyword generation system uses the PA swarm algorithm to generate honeywords using three methods:

Alphabet characters

Random digits

Special characters.

The PA algorithm finds solutions based on proximity, quality, diversity, stability, and adaptability, making it suitable for creating alphabet honeywords.

It is worth noting at this stage that the PA is configured to operate discretely, meaning it will process non-numeric inputs. These inputs will consist of letters, symbols, and numbers. Because the algorithm's addition and multiplication operations effectively exchange letter positions in passwords, the generated passwords vary accordingly, satisfying the required conditions and achieving flatness. In this context, mathematical operations (addition, multiplication, and subtraction) correspond to insertion, deletion, and exchange operations [17, 18]. Using this mapping, the PA was adapted for honeyword generation. In a traditional user interface, users are required to create accounts with a username and a password containing letters, digits, and special characters. The system generates 49 honeywords, giving an attacker a 1 in 49 chance (about 2%) of selecting a sugar word. The sugar word remains difficult to guess even if a token is known. Each token is repeated seven times, providing a 14% chance of guessing the sugar word. In other words, the password has three parts (Letters, Numbers & Special Characters). At each time, the approach will generate one type.

Example 1:

Real Password = "flower1234@@"

At 1st Seven Times change the letters, the generated honeywords="floor1234@@", "flood1234@@", "floozy1234@@", "rose1234@@", "bloom1234@@", "floom1234@@", "fool1234@@"

At 2nd Seven Times change the numbers, the generated honeywords="floor1111@@", "flood4321@@", "floozy0123@@", "rose7890@@", "bloom4444@@", "floom2468@@", "fool1356@@"

At 3rd Seven Times change the Special Chars., the generated honeywords="floor1234! #", "flood1234##", "floozy1234%$", "rose1234&%", "bloom1234&@", "floom1234#!", "fool1234&&"

The proposed system aims to improve honeyword generation, enhance features, and resolve issues from previous methods. It follows six parallel steps for creating honeywords from the sugar word.

Step 1: Separate Characters

Break the "sugar word" into letters, digits, and special characters. Keep any token that matches the username unchanged.

Step 2: Generate Letters

Use the PA alphabet generator to create new letters from the letter tokens.

Step 3: Generate Digits

Check digit tokens against important numbers; if no match is found, generate a random digit, otherwise randomly select one from the list.

Step 4: Special Characters Generation

Use a random generator for a special character for the tokens.

Step 5: Collect Honeywords

Gather all generated honeyword tokens.

Step 6: Create Sweetwords

Combine sugar and honey words to produce 49 "sweet words," mix their order, and send the user index (u(i)) and sugar word index (c(i)) to the honey checker. Hash and store the sweet words without saving the sugar word index.

6. THE PROPOSED ALGORITHM FOR GENERATING THE TOKENS OF THE HONEYWORD

The suggested structure has utilized three algorithms to generate the tokens of the honeyword. In addition, the proposed PA is used to generate the honeyword token solutions. The system consists of three parallel components: character generation using the PA, and digit and special-character generation using simple random procedures.

6.1 The proposed Pelican Algorithm

This segment of the honeyword is crucial because it is a key focus for attackers trying to discover the real password in in Figure 1. The PA acts as a generator of random digits. It uses a random-digit generation token derived from the sugar word as the seed to produce random digits for the honeyword algorithm. It outlines the basic steps of the PA for generating these random digits. This section explains the reasoning behind the development of the PA algorithm and describes its mathematical basis.

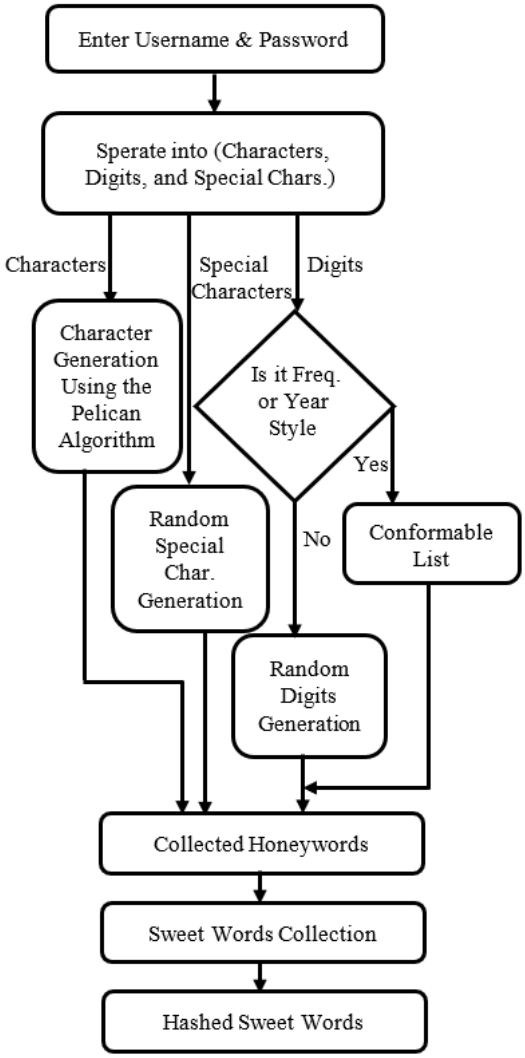

Figure 1. The proposed system block diagram

Algorithm 1. The PA

The steps of the proposed PA for the random-digit generation token are as follows:

Begin
Input the sugar_word (Real Password);
Generate N Honeywords (Pelican population) as neighbors from sugar_word;
Calculate the words' similarity distance and use it as the objective function;
For t = 1: T
Generate a random prey position.
For I = 1: N
Phase 1: (exploration phase) moving towards prey.
For j = 1: m
Use Eq. (4) to calculate the jth dimension of the new status
End.
Using Eq. (5), update the ith population member.
Phase 2: (exploitation phase) water surface winging.
For j = 1: m.
Using Eq. (6), calculate the jth dimension of the new status.
End.
Using Eq. (7), update the ith population member.
End.
Based on the words' similarity distance, update the best candidate solutions.
End.
Output the best candidate solutions based on the words' similarity distance.
End POA.

The Alphabetic Style characters present a problem that requires applying operations to generate neighbors using the pelican equations. For each token, four operations will be performed:

a) Insertion: Add random characters at random positions.
b) Dropping: Remove characters from random positions.
c) Translocation: Swap two characters' positions.
d) Swapping: Exchange two characters with random others.
The best new token will replace the original one.

6.2 The generator of the tokens of the random digits

The algorithm will start with a digit from the word "sugar" and generate six tokens of the same length. Each token will be repeated seven times, resulting in 42 tokens organized into six groups. Seven copies of the original digit seed will be added, totaling 49 tokens.

Algorithm 2

The digit tokens' generic steps generating algorithm includes the following:

a. Set d to be the generated digit tokens.
b. Use randomly chosen digits to change the digits dp of the digit seed.

6.3 The generator of the tokens of the random special characters

The algorithm generates special characters from a seed word, creating six tokens of the same length. Each of these tokens gets replicated seven times, resulting in 42 tokens organized into six groups. Digit seed will also be included in seven copies, totaling 49 special character tokens.

Algorithm 3. Generating Special Character Tokens:

a) Decide the number of tokens (s).
b) Randomly replace special characters in the seed (sp).

Example 2: The seed token is "letter1979?", for the sugar_word "letter1979?", generating "11" as the digits and "?" as the special character. The generated tokens are connected by the system like this.

The system will randomly shuffle the sweet words, sending the user's index (u(i)) and the sugar word index (c(i)) to the honey checker. The sweet words will then be hashed and stored (Table 1).

6.4 The proposed evaluation criteria

This method, called Digit Randomly Tokens, evaluates generated solutions using an objective function that sums several criteria. These tokens rely on a seed token from a "sugar_word." A new measure introduced is the "approximation factor," which ranges from 0 to 1 and considers four weighted criteria [13, 20]:

1. Character Similarity: Compares the character similarity between the generated tokens and the seed.
2. Length Similarity: Assesses the length similarity of both tokens.
3. Similarity of Part of Speech (POS): Evaluates the similarity of their parts of speech.
4. Meaningfulness: Checks if the generated token is a valid English word.

Table 1. A sample of the generated tokens

Generated Tokens						
letter1983+	latter1979-	gitter1980#	fitter1976?	litter1988-	inters1980%	fisher1975$
letter1971!	latter1977+	gitter1977!	fitter1987@	litter1972+	inters1971@	fisher1985!
letter1972@	latter1983#	gitter1982@	fitter1975#	litter1978#	inters1982!	fisher1972@
letter1980$	latter1981$	gitter1977$	fitter1979&	litter1971@	inters1977#	fisher1983#
letter1982%	latter1978@	gitter1983+	fitter1982$	litter1980%	inters1983*	fisher1973*
letter1979?	latter1975&	gitter1981-	fitter1977*	litter1981&	inters1974?	fisher1972%
letter1977&	latter1980?	gitter1976?	fitter1975!	litter1977*	inters1981%	fisher1980&

6.5 The findings and discussions

Table 2 shows the values of the parameters, the experiments' outcomes, and an evaluation of the proposed honeyword generation technique and earlier approaches. The proposed system uses several significant parameters that impact its efficiency.

Table 2. The proposed system parameters range

Parameters	Range
Population Size (Pelican Numbers)	25 – 100
Generated digits tokens number	4 – 6
Number of the generated alphabet tokens	5 – 8
Max Generation	20 – 50
Generated special characters tokens Number	3- 5

To further validate the proposed system's flattening property, a security interpretation based on the honeyword distribution is provided. For each sugar word, the system generates 49 tokens, including one real password and 48 honeywords, all constructed to exhibit comparable similarity characteristics. Because each token is indistinguishable in structure and likelihood, an attacker selecting a token at random has a success probability of 1/49 (about 2%), which corresponds to ideal flatness. Furthermore, even if an attacker obtains partial knowledge of a token group, each candidate appears with equal frequency, limiting the probability of identifying the correct sugar word to 1/7 (about 14%). This analytical evaluation confirms that the proposed method achieves unconditional flatness without restrictive assumptions, in contrast to earlier honeyword generation techniques.

7. EXPERIMENTAL RESULTS

The experimental findings are categorized into two sections. The initial section distinguishes letters, digits, and special symbols, whereas the second section analyzes the entire password. The system was tested with various password tokens, which are crucial since attackers attempt to guess passwords. This section presents results for honeyword tokens generated by the system, each containing ten letters, three numbers, and two special characters. The initial eight-letter tokens consist of two real words and two meaningless ones. The study used a population size (Pop-size) ranging from 25 to 100 pelicans and a maximum generation (Max-gen) between 20 and 50 for letter tokens. Testing was conducted with four population sizes: 25, 50, 75, and 100, and a maximum generation of 40. All Pop-size values performed well, with a Pop-size of 100 yielding the best approximation factor. The algorithm produces the five best tokens with their approximation factors. Tokens with numbers and special characters are generated randomly while maintaining the seed token's length, resulting in a final output of six tokens.

Regarding the evaluation of the results, the focus is primarily on the flattening factor, as it is the main factor that prevents an attacker from detecting the real (correct) password, and on the similarity between the generated honeywords and the original (real) password.

Table 3 shows the top five honeywords produced by Algorithm 1 for different population sizes, selected according to the approximation factor defined in Section 6.4.

7.1 A comparison study

The proposed system demonstrates that the novel PA outperforms traditional honeyword generation methods. This section evaluates PA against previous approaches across three key aspects: enhancing the overall honeyword generation process, improving the quality and characteristics of generated honeywords, and addressing the limitations of earlier techniques (see Table 4). These comparisons highlight how PA not only produces more diverse and secure honeywords but also resolves inefficiencies present in prior methods.

A. Improving the Honeyword-Generating Process.

The PA method uniquely generates honeywords by creating password tokens from a seed token, which are then transformed into honeyword tokens.

In contrast, earlier approaches typically relied on minor modifications of existing passwords, suggested honeywords, required user-provided honeywords, used passwords from other users, generated honeywords from publicly available password lists, or incorporated personal information. This distinction highlights PA's more systematic and secure approach to honeyword generation.

B. Honeyword Features.

The proposed PA improves honeyword features that earlier methods often lack. The key honey_word features are as follows.

Flatness represents the probability that an attacker correctly identifies the real password among multiple fake options. Ideally, a method achieves perfect flatness of 1/k, keeping the chance of a correct guess very low. Unlike earlier approaches, the proposed PA system attains perfect flatness without requiring specific conditions. For comparison, the original Juels–Rivest honeyword scheme had a 5% chance of an attacker guessing the correct password [26]. The new system reduces the probability to approximately 2% (1/49). In previous systems, knowing one token allowed attackers to guess the sugar word, but in the proposed system, even if an attacker knows a token, guessing the correct sugar word remains difficult, with only 14% (1/7) of selecting the correct one, as each token appears seven times.

Table 3. Experimental results of the proposed system

No.	Token	Pelican Size	Honeyword Token / Approximation Factor				
			Honeyword 1	**Honeyword 2**	**Honeyword 3**	**Honeyword 4**	**Honeyword 5**
1	flood	25	floor/0.96	flower /0.81	flown / 0.81	floer / 0.78	blader / 0.75
		50	floor/ 0.96	booker /0.78	wooder/0.81	roomer/0.81	flower /0.81
		75	bloor/0.96	fodder /0.78	bloom / 0.96	flawed/0.78	wooder/0.81
		100	gloor / 0.96	bloom/ 0.96	blower /0.86	booker /0.81	roomer/0.81
2	live	25	wive / 0.95	love / 0.95	olive / 0.88	nive / 0.9	mice / 0.88
		50	wive / 0.95	olive / 0.88	five / 0.95	wide / 0.81	love / 0.95
		75	hive / 0.95	love / 0.95	lice / 0.95	olive / 0.88	five / 0.95
		100	like / 0.95	life / 0.95	lime / 0.95	line / 0.95	love / 0.95
3	cake	25	fake / 0.95	male / 0.88	call/0.81	take/0.91	came/0.78
		50	lake / 0.95	like/0.95	hake/0.95	case/0.95	cache/0.81
		75	fake / 0.95	lame/0.95	cache/0.81	mace/0.95	nake/0.94
		100	make / 0.95	sake / 0.95	lake / 0.95	lame/0.95	cane/0.95
4	rocker	100	docker/0.95	locker/0.95	rockey/0.95	rocket/0.95	locket/0.91
5	dice	100	lice / 0.95	nice / 0.95	mice / 0.95	dike / 0.9	dime / 0.95
6	sunrise	100	sunshine/0.7	sunset/0.8	sunroof/0.7	sunblock/0.8	sunbath/0.8
7	coffee	100	cofferet/0.86	coffer/0.96	coffine/0.91	coffle/0.88	cofferer/0.78
8	duck	100	dick/0.96	dock/0.96	luck/0.96	duct/0.96	docker//0.81
9	cvcbmw	100	zvzbmw/0.88	asdbmw/0.8	nubmw/0.9	cvcbnm/0.91	cvcbmn//0.96
10	qwerty	100	qwerzy/0.91	qwerds/0.88	qwergh/0.81	swertg/0.81	werety/0.78
11	1979	100	1980	1978	1981	1977	1986
12	1234	100	1122	1233	1245	1133	1213
13	2468	100	2411	2456	8642	8264	2131
14	@#!	100	#$@	#@$	&%$	&@#	@$&
15	&%@	100	%&@	@$&	$#$	#%&	#!%

Table 4. The most significant honeyword features

No	Techniques	DoS Resistance	Flatness	Storage
1	Proposed Pelican Algorithm	Strong	Unconditionally	No extra storage cost
2	Chaffing-by –tweaking-digits [14]	Weak	Conditionally	No extra storage cost
3	Chaffing-by-tail-tweaking [14]	Weak	Conditionally	No extra storage cost
4	Modeling syntax [14]	Strong	Conditionally	No extra storage cost
5	Take-a-tail [14]	Strong	Unconditionally	No extra storage cost
6	Simple model [14]	Strong	Conditionally	No extra storage cost
7	Chaffing with “tough nuts” [14]	Strong	N/A	extra storage cost
8	Random pick [14]	Strong	Conditionally	No extra storage cost
9	Storage-index [16]	Weak	Conditionally	extra storage cost
10	Hybrid generation methods [14]	Strong	Conditionally	No extra storage cost
11	Evolving password model [17]	Strong	Conditionally	No extra storage cost
12	Paired Distance Protocol [15]	Strong	Conditionally	extra storage cost
13	Append-secret model [17]	Strong	Conditionally	No extra storage cost
14	User-profile model [17]	Weak	Conditionally	extra storage cost
15	Dictionary attack method [18]	Weak	Conditionally	No extra storage cost
16	User information method [18]	Weak	Conditionally	extra storage cost
17	Shuffling characters method [19]	Weak	Conditionally	No extra storage cost
18	Generic password list method [20]	Strong	Conditionally	No extra storage cost

Flatness: This measures the likelihood of someone successfully guessing the correct password among several fake ones, achieving a flatness of $(\frac{1}{k})$ under certain conditions. The new password system (PA) achieves perfect flatness without any conditions, a significant advantage over older systems that only achieve this under specific situations.

In the original honeyword system by Juels and Rivest [26], the chance of guessing the correct password was 5% (1/20), while the proposed system reduces this to about 2% (1/49). Additionally, in older systems, a hacker could easily guess the correct password if they knew one token. In contrast, in the proposed system, each token is repeated seven times, giving

only about a 14% chance (1/7) of guessing the correct password, even if one token is known.

- **DoS resistance**: The system generates hard-to-guess honeywords. When a honeyword is entered, it triggers a denial of service (DoS) attack, disrupting system services. This method ensures the honeywords are not guessable by attackers, even if they know the real password, the sugar word. Previous methods often produced guessable honeywords.
- **Storage**: Unlike older methods that store extra details, the new password app only keeps usernames and simple words.

Table 5 compares the suggested PA and earlier generation techniques concerning the key attributes of honeywords. Addressing the problems of past approaches. Previous methods for generating honeywords encountered numerous challenges. The suggested PA resolves the seven key problems associated with honeyword systems. These seven challenges are listed below:

- **Conditional Flatness Issue**: Achieving perfect flatness needs specific conditions, which is a limitation. Unconditional flatness, a strength, requires no conditions. Unlike many older methods, the proposed Password Architecture (PA) guarantees unconditional perfect flatness.
- **Weak DoS Resistance Issue**: An attacker could guess honeywords, indicating weak Denial-of-Service (DoS) resistance. The PA demonstrates strong DoS resistance, whereas older methods often fall short.
- **Storage Overhead Issue**: Some methods incur extra storage costs, but the proposed PA has none.
- **Correlation Issue**: Connections between password and username can help identify the correct password. The PA mitigates this by maintaining related elements in the honey_words and, hence, securing them.
- **Consecutive or Frequent Numbers Issue**: Operators frequently choose an easy sequence of number patterns for passwords, risking exposure of the sugar word. The PA addresses this by randomly selecting numbers from common patterns.
- **Special Date Issue**: Users often use significant dates in passwords. The system prevents this by randomly selecting years from a list of the last 50 years if such digits appear in the sugar word.
- **User Information Security Issue**: Many older methods use personal knowledge-based questions, which pose security risks if the data is compromised. The PA strengthens security by not requiring personal information.

Table 5. The key honeyword system issues

#	Methods	Conditional Flatness Issue	Storage Overhead Issue	Correlation Issue	Special Date Issue	Weak Denial-of-Service (DoS) Resistance Issue	Consecutive and Frequent Numbers Issue	User Information Security Issue
1	Proposed Pelican Algorithm	No	No	No	No	No	No	No
2	Chaffing-by–tweaking-digits [14]	Yes	No	Yes	Yes	Yes	Yes	No
3	Chaffing-by-tail tweaking [14]	Yes	No	Yes	Yes	Yes	Yes	No
4	Modeling syntax [14]	Yes	No	Yes	Yes	No	Yes	No
5	Simple model [14]	Yes	No	Yes	No	No	No	No
6	Take-a-tail [14]	No	No	No	No	No	No	No
7	Chaffing with “tough nuts” [14]	N/A	Yes	No	N/A	No	N/A	No
8	Hybrid generation methods [14]	Yes	No	Yes	Yes	No	Yes	No
9	Random pick [14]	Yes	No	Yes	No	No	No	No
10	Paired Distance Protocol [15]	Yes	Yes	No	No	No	Yes	No
11	Storage-index [16]	Yes	Yes	Yes	No	Yes	No	No
12	User-profile model [17]	Yes	Yes	Yes	No	Yes	Yes	Yes
13	Evolving password model [17]	Yes	No	Yes	Yes	No	Yes	No
14	User information method [18]	Yes	Yes	Yes	No	Yes	Yes	Yes
15	Append-secret model [17]	Yes	No	No	No	No	Yes	No
16	Generic password list method [20]	Yes	No	Yes	No	No	No	No
17	Dictionary attack method [18]	Yes	No	Yes	No	Yes	No	No
18	Shuffling characters method [19]	Yes	No	Yes	Yes	Yes	Yes	No

8. CONCLUSIONS

This study proposed a novel honeyword generation method based on the PA. The method decomposes passwords into letters, digits, and special characters and generates honeywords using a token-based approach. Experimental results show a significant improvement in flatness (about 2%, 1 in 49) compared with the original Juels-Rivest scheme (about 5%, 1 in 20), and the guessing probability was reduced to about 14% (1 in 7).

Despite these improvements, the proposed method has limitations. Its performance may decline for completely random or non-semantic passwords, where generating meaningful honeywords is inherently challenging. Additionally, metaheuristic optimization introduces computational overhead that may affect scalability in large systems.

Future work will address these limitations by exploring alternative optimization methods, optimizing runtime through parallel processing, and evaluating the method on larger and more diverse password datasets. Further research could also investigate adaptive honeyword generation based on password complexity and user behavior to enhance security while maintaining usability.

REFERENCES

[1] Genc, Z.A., Kardaş, S., Kiraz, M.S. (2017). Examination of a new defense mechanism: Honeywords. In IFIP International Conference on Information Security Theory and Practice, Crete, Greece, pp. 130-139. https://doi.org/10.1007/978-3-319-93524-9_8

[2] Kusuma, A.B., Pramadi, Y.R. (2019). Implementation of honeywords as a codeigniter library for a solution to password-cracking detection. IOP Conference Series: Materials Science and Engineering, 508(1): 012134. https://doi.org/10.1088/1757-899X/508/1/012134

[3] Win, T., Moe, K.S.M. (2018). Protecting private data using improved honey encryption and honeywords generation algorithm. Advances in Science, Technology and Engineering Systems Journal, 3(5): 311-320.

[4] Palaniappan, S., Parthipan, V., Stewart kirubakaran, S., Johnson, R. (2018). Secure user authentication using honeywords. In International conference on Computer Networks, Big data and IoT, Madurai, India, pp. 896-903. https://doi.org/10.1007/978-3-030-24643-3_105

[5] Erguler, I. (2014). Some remarks on honeyword based password-cracking detection. Cryptology ePrint Archive.

[6] Tezel, B.T., Mert, A. (2021). A cooperative system for metaheuristic algorithms. Expert Systems with Applications, 165: 113976. https://doi.org/10.1016/j.eswa.2020.113976

[7] Yasser, Y.A., Sadiq, A.T., AlHamdani, W. (2023). Honeyword generation using a proposed discrete salp swarm algorithm. Baghdad Science Journal, 20(2): 16. https://doi.org/10.21123/bsj.2022.6930

[8] Malik, H., Iqbal, A., Joshi, P., Agrawal, S., Bakhsh, F.I. (2021). Metaheuristic and evolutionary computation: Algorithms and applications. In Studies in Computational Intelligence. https://doi.org/10.1007/978-981-15-7571-6

[9] Green, D., Aleti, A., Garcia, J. (2017). The nature of nature: Why nature-inspired algorithms work. In Nature-Inspired Computing and Optimization: Theory and Applications, pp. 1-27. https://doi.org/10.1007/978-3-319-50920-4_1

[10] Mohammed, A.A., Abdul-Hassan, A.K., Mahdi, B.S. (2019). Authentication system based on hand writing recognition. In 2019 2nd Scientific Conference of Computer Sciences (SCCS), Baghdad, Iraq, pp. 138-142. https://doi.org/10.1109/SCCS.2019.8852594

[11] Jo, H.J., Yoon, J.W. (2015). A new countermeasure against brute-force attacks that use high performance computers for big data analysis. International Journal of Distributed Sensor Networks, 11(6): 406915. https://doi.org/10.1155/2015/406915

[12] Genç, Z.A., Lenzini, G., Ryan, P., Vazquez Sandoval, I. (2018). A security analysis, and a fix, of a code-corrupted honeywords system. In 4th International Conference on Information Systems Security and Privacy. https://doi.org/10.5220/0006609100830095

[13] Alper, Z., Lenzini, G., Ryan, P.Y.A., Vazquez Sandoval, I. (2019). A critical security analysis of the password-based authentication honeywords system under code-corruption attack. Information Systems Security and Privacy: 4th International Conferences (ICISSP 2018), Funchal-Madeira, Portugal, 4: 125-151.

[14] Erguler, I. (2015). Achieving flatness: Selecting the honeywords from existing user passwords. IEEE Transactions on Dependable and Secure Computing, 13(2): 284-295. https://doi.org/10.1109/TDSC.2015.2406707

[15] Chakraborty, N., Mondal, S. (2017). On designing a modified-UI based honeyword generation approach for overcoming the existing limitations. Computers & Security, 66: 155-168. https://doi.org/10.1016/j.cose.2017.01.011

[16] 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. https://doi.org/10.1109/TDSC.2018.2824323

[17] 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. In SAI Global Conference on Innovation and Technology (SAI). https://doi.org/10.14569/IJACSA.2019.0100305

[18] Catuogno, L., Castiglione, A., Palmieri, F. (2015). A honeypot system with honeyword-driven fake interactive sessions. In 2015 International Conference on High Performance Computing & Simulation (HPCS), Amsterdam, Netherlands, pp. 187-194. https://doi.org/10.1109/HPCSim.2015.7237039

[19] Yasser, Y.A., Sadiq, A.T., AlHamdani, W. (2022). A proposal for honeyword generation via meerkat clan algorithm. Cybernetics and Information Technologies, 22(1): 40-59. https://doi.org/10.2478/cait-2022-0003

[20] Brindtha, J., Hithaeishini, K.R., Komala, R., Abirami, G., Arul, U. (2017). Identification and detecting of attacker in a purchase portal using honeywords. In 2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM), Chennai, India, pp. 389-393. https://doi.org/10.1109/ICONSTEM.2017.8261414

[21] Nathezhtha, T., Vaidehi, V. (2017). Honeyword with salt-chlorine generator to enhance security of cloud user credentials. In International Symposium on Security in

Computing and Communication, Manipal, India, pp. 159-169. https://doi.org/10.1007/978-981-10-6898-0_13
[22] Hadi, R.M., Abdullah, S.H., Abedi, W.M.S. (2022). Proposed neural intrusion detection system to detect denial of service attacks in MANETs. Periodicals of Engineering and Natural Sciences, 10(3): 70-78. https://doi.org/10.21533/pen.v10.i3.644
[23] Abdullah, S.H., Abedi, W.M.S., Hadi, R.M. (2022). Enhanced feature selection algorithm for pneumonia detection. Periodicals of Engineering and Natural Sciences, 10(6): 168-180.
[24] Bozeman, J. (2015). Cyber security essentials. Control Engineering, 62(1): 90-92.
[25] Moe, K.S.M., Win, T. (2017). Improved hashing and honey-based stronger password prevention against brute force attack. In 2017 International Symposium on Electronics and Smart Devices (ISESD), Yogyakarta, Indonesia, pp. 1-5. https://doi.org/10.1109/ISESD.2017.8253295
[26] Juels, A., Rivest, R.L. (2013). Honeywords: Making password-cracking detectable. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, pp. 145-160. https://doi.org/10.1145/2508859.2516671