


DOI: <http://dx.doi.org/10.21123/bsj.2022.6930>

## Honeyword Generation Using a Proposed Discrete Salp Swarm Algorithm

Yasser A. Yasser<sup>1\*</sup> 

Ahmed T. Sadiq<sup>1</sup> 

Wasim AlHamdani<sup>2</sup> 

<sup>1</sup>Computer Science Department, University of Technology, Baghdad, Iraq.

<sup>2</sup>Information Technology Department, University of the Cumberland, KY, USA.

\*Corresponding author: [cs.19.28@grad.uotechnology.edu.iq](mailto:cs.19.28@grad.uotechnology.edu.iq)

E-mail addresses: [Ahmed.T.Sadiq@uotechnology.edu.iq](mailto:Ahmed.T.Sadiq@uotechnology.edu.iq), [wasim.alhamdani@ucumberlands.edu](mailto:wasim.alhamdani@ucumberlands.edu)

Received 14/1/2022, Revised 2/5/2022, Accepted 4/5/2022, Published Online First 20/9/2022  
Published 1/4/2023



This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).

### Abstract:

Honeywords are fake passwords that serve as an accompaniment to the real password, which is called a “sugarword.” The honeyword system is an effective password cracking detection system designed to easily detect password cracking in order to improve the security of hashed passwords. For every user, the password file of the honeyword system will have one real hashed password accompanied by numerous fake hashed passwords. If an intruder steals the password file from the system and successfully cracks the passwords while attempting to log in to users’ accounts, the honeyword system will detect this attempt through the honeychecker. A honeychecker is an auxiliary server that distinguishes the real password from the fake passwords and triggers an alarm if intruder signs in using a honeyword. Many honeyword generation approaches have been proposed by previous research, all with limitations to their honeyword generation processes, limited success in providing all required honeyword features, and susceptibility to many honeyword issues. This work will present a novel honeyword generation method that uses a proposed discrete salp swarm algorithm. The salp swarm algorithm (SSA) is a bio-inspired metaheuristic optimization algorithm that imitates the swarming behavior of salps in their natural environment. SSA has been used to solve a variety of optimization problems. The presented honeyword generation method will improve the generation process, improve honeyword features, and overcome the issues of previous techniques. This study will demonstrate numerous previous honeyword generating strategies, describe the proposed methodology, examine the experimental results, and compare the new honeyword production method to those proposed in previous research.

**Keywords:** Authentication, Honeyword, Password, Salp algorithm, Swarm algorithm.

### Introduction:

Password-based authentication is the most widely used authentication mechanism due to its simplicity and memorability. However, this strategy has been targeted by bad actors using a variety of attack methods, including password cracking.<sup>1,2</sup> Password cracking is a dangerous and typically illegal method of recovering passwords from computer information stored on or sent through a device<sup>3</sup>.

Honeywords provide a convenient method for increasing the number of fake passwords associated with each user’s account, thus improving the safety of hashed passwords and making password cracking less difficult to detect<sup>4,5</sup>. An intruder who gains access to a database of hashed passwords and flips the hash will not be able to retrieve the original password. Instead, a “quiet alarm” will be issued if

a honeyword is used during the login procedure<sup>6</sup>. Honeychecker is a secondary server that can differentiate between the original password and honeywords and is connected to the login server over a secure connection.<sup>7,8</sup>

A metaheuristic is a method or heuristic used in computer science and mathematical optimization to find, create, or choose a heuristic that may provide the best available solution to a problem<sup>9,10</sup>. An optimization problem is a problem in mathematics, computer science, or economics in which the aim is to find the best solution from a set of alternatives<sup>11</sup>.

Swarm intelligence algorithms depend on the relationships between live creatures; inspiration for the algorithms is generally derived from nature, particularly biological systems<sup>12</sup>. Although there is

no centralized control structure prescribing how individual agents should behave, local—and, to some extent, random—interactions between such agents lead to the creation of intelligent global behavior<sup>13</sup>.

The salp swarm algorithm (SSA) is a swarm intelligence algorithm influenced by biology, and functions as a metaheuristic optimization method that mimics the natural swarming behavior of salps<sup>14,15</sup>.

Many honeyword generation techniques have been presented by previous research. However, all previous proposals have included flaws in the generation process, been unable to provide all the necessary honeyword characteristics, and exhibited many honeyword issues. The salp swarm algorithm (SSA) proposed in this work is a bio-inspired metaheuristic intelligence algorithm that functions as a novel method for honeyword generation. The proposed method presents a discrete salp swarm algorithm to benefit from its ability to obtain excellent solutions, successfully enhance random solutions, converge toward the optimum solutions, and balance exploration and exploitation. This method will enhance the generation process, provide all the required honeyword characteristics, and overcome the shortcomings of earlier approaches.

The many contributions presented by this research include the following:

- The proposed system employs the salp swarm algorithm to generate honeywords, which is a unique strategy.
- The proposed generation strategy will improve the honeyword creation process, support honeyword features, and address the limitations of previous techniques.
- The proposed generation algorithm's password alphabet token produced excellent results for creating meaningful words from meaningful words; however, the most promising aspect is the proposed algorithm's ability to identify meaningful words from rubbish words.
- The approximation factor is suggested by the proposed technique as an assessment criterion for the produced honeyword (alphabet token).
- Using the proposed method, the sugarword cannot be predicted even if the attacker knows one of the sugarword tokens. Every token is redundant five times in sweetwords. Thus, if one of the sugarword tokens is known by the attacker, the probability of obtaining the sugarword at random is (1/5=20%).

The specific terms used throughout this paper in the context of the honeyword system are defined as follows<sup>16</sup>:

- Sweetwords: These collect both the real password (sugarword) and the fake passwords (honeywords) (k).
- Honeywords: The fake passwords generated by the honeyword system (k-1).
- Sugarword: The real password provided by the user.
- Honeychecker: An auxiliary server that can distinguish between the sugarword and the honeywords. The honeychecker is linked to the login server through a secure connection and is responsible for triggering a silent alarm if a honeyword is used during the login process to declare that a breach may happen.

The remainder of this paper will examine previous research in the field of honeyword generation, provide a simple illustration of the honeyword process, demonstrate the salp swarm algorithm, explain the proposed system with the suggested discrete SSA, present the experimental results, compare the suggested method with previous honeyword generation approaches, discuss the implications of this research, and present the research conclusion.

#### Literature Review:

Honeyword generating methods have been the subject of extensive research in recent years, with much convergent study in this field. The following will examine a key selection of such research.

- In<sup>16</sup> (2013), Juels and Rivest:

Many honeyword creation strategies are suggested in this study, including changing a piece of the password, using a dictionary, adding a tail from the system, using honeywords given by the system, using honeywords supplied by the user, and hybrid approaches. These methods are grouped into two categories based on whether or not they influence the user interface (UI), with each category containing several honeyword creation techniques. The two categories are:

##### 1. Legacy-UI

(Chaffing-by-tail-tweaking; Chaffing-by-tweaking-digits; Simple model; Modeling syntax; "Tough nuts;" Hybrid generation methods).

##### 2. Modified-UI

(Take-a-tail; Random pick).

- In<sup>17</sup> (2015), Ergular:

The "Storage-index" technique proposed in this study presents an alternative method for honeyword creation that picks honeywords based on current user passwords in the system in order to generate realistic honeywords. Honeywords are used in the recommended strategy for detecting

password cracking. However, instead of producing honeywords and preserving them in a password database, this method mimics honeywords using existing passwords.

- In <sup>18</sup> (2017), Chakraborty and Mondal:

This study proposes the Paired Distance Protocol (PDP) as a new honeyword generation mechanism with a new user interface. A user must enter three pieces of information to log in: a username, a password, and a password-tail. When enrolling, the user selects a password-tail of  $t > 1$  from a selection of 1. alphabetic characters (a-z), and 2. digits in addition to the username and password (0–9).

- In <sup>19</sup> (2018), Akshima et al.:

This study offers the “evolving-password model,” the “user-profile model,” and the “append-secret model” as new honeyword generation methodologies.

- Evolving-password model: To complete the procedure, this model uses two different calculation steps. First, tracking the number of times that password patterns and tokens have been used; second, creating honeywords and keeping frequency tables from previous frequencies.
- User-profile model: In this model, honeywords are created by combining various user profile data and creating separate sets from given data that comprise tokens of various sorts, such as “alphabet-strings,” “digit-strings,” and “special-character-strings.”
- Append-secret model: The system used in this model asks for the user’s login, password, and an optional item, such as  $e$ , to generate a random string  $s$  that contains numbers, letters, and symbols. After performing the function  $f(p | e | s)$ , the model returns  $r$ .  $H(\text{password} | r)$  will be saved in the password file of the system.
- In <sup>20</sup> (2019), Akif et al.:

This study proposes a new honeyword creation strategy that incorporates all four methods. As a result, the system acquires four groups of honeywords generated from the following sources:

1. Existing user information: Data is generated using two-part personal questions. The first portion will be about letters, and the second will be about numbers. The answers to the first and second parts will be combined to generate honeywords.
2. A dictionary attack: Using the real password with a change of up to three digits or characters is the basic principle behind producing acceptable honeywords after scanning via the dictionary attack.
3. A generic password list: This honeyword group is made up of honeywords chosen at random

from a database of the 500 worst passwords. The worst passwords mean they are easier to guess.

4. Shuffling the characters: Scrambled characters or numerals from the ID user are combined; that is, the honeyword is created by mixing scrambled characters or digits from the ID user.

### Honeywords

The honeywords technique works by generating honeywords (fake passwords), adding a sugarword (true password), then hashing and inserting them all as sweetwords into the username and password database <sup>21,22</sup>. If the intruder obtains plain passwords from hashed passwords, the intruder must then successfully guess the true password from among the sweetwords; otherwise, a silent alarm is triggered to the system administrator, indicating that password cracking may be occurring <sup>23,24</sup>. Administrator responses to this alarm are defined by the organization’s policy and may include suspending, blocking, or alerting the compromised account. <sup>25</sup>

For flatness, let  $x$  be the intruder’s assessed probability of correctly anticipating the sugarword. Because an intruder can estimate the sugarword randomly with a  $1/k$  likelihood of success, the intruder has at least a  $(1-(1/k))$  chance of selecting a honeyword if the honeywords are as flat as possible <sup>26,27</sup>.

When a user wants to log in to an account, the login server checks the user’s input username and password. If the password is incorrect, the system asks the user to try again. If the password matches one of the sweetwords, the system submits it to the honeychecker for verification. An alert is triggered if the password matches one of the honeywords. If it matches the sugarword, then it allows the user to log into the account <sup>28,29</sup>.

### Salp Swarm Algorithm SSA

A salp is a type of marine creature that is categorized as part of the Salpidae family. It has a barrel shape with holes at the end—similar to a jellyfish—that pumps water across their bodies, allowing them to move and eat through inner feeding filters <sup>30</sup>. Many behaviors, such as swarming, are shared by marine creatures. Swarming behavior is one of the most intriguing characteristics of salps. Salps frequently form a swarm, known as a salp chain, in deep waters <sup>31</sup>. Although the exact rationale for this activity is unknown, some researchers believe it is done to improve locomotion and foraging. The salp chain can be divided into various sub-chains, each of

which may take a different route to achieve the same aim. Each sub-chain will have its own leader and followers<sup>32</sup>.

Thus, a salp swarm algorithm (SSA) is an optimization metaheuristic swarm algorithm inspired by this biological behavior. An SSA is designed to imitate the swarming behavior of salps in their natural environment<sup>33</sup>. The SSA's behavior mimics the behavior of a natural salp chain seeking optimal feeding sources. The chain salp in an SSA is divided into two groups based on the locations of the individuals (salps) in the chain: leaders and followers. The leader salp is the first in a chain of salps, while the remaining salps are termed followers; the individual (leader) guides the movements of the others (followers)<sup>34,35</sup>.

Algorithm 1. shows the steps of SSA that begin by generating the initial population of salps considering the lower and upper bounds of the positions of the leader and the source of food. Calculate the fitness of salps, determining the best salp as a leader and the rest as followers. Let the best salp as a source of food. Eq. 1, is used to calculate  $v_1$ , which is a parameter that gradually decreases across iterations to balance exploration and exploitation. The leader position is updated by Eq. 2, and the followers by Eq.3. Adjust the salps that have been outside the search space according to the lower and upper bounds. Repeat the algorithm until the end condition is satisfied, then return the best salp as the solution<sup>36-38</sup>.

**Algorithm 1. The general steps of the Salp Swarm algorithm<sup>36-38</sup>.**

Step 1: Set up the algorithm parameters ( $N$  (pop-size),  $S$  (source of food),  $u$  (upper-bound),  $l$  (lower-bound)).  
 Step 2: Generate  $N$  of salps randomly, considering  $u$  and  $l$ .  
 Step 3: Moving of salps towards  $S$   
     a: Compute the fitness of salps, takes the best salp as a leader and the rest as followers.  
     b:  $S =$  Best salp  
     c: Compute  $v_1$  by Equation 1  
     d: For every salp of  $N$   
         If the salp is a leader, update its position by Equation 2.  
         If the salp is a follower, update its position by Equation 3.  
     e: Adjust the salps according to  $u$  and  $l$ .  
 Step 4: Repeat Step 3 until the end condition is satisfied.  
 Step 6: Return  $S$ .

Equation 1 computes the  $v_1$  parameter, which is responsible for the balance between exploration and exploitation, where  $T$  is the maximum number of iterations and  $t$  is the current iteration of<sup>36-38</sup>.

$$v_1 = 2e^{-\left(\frac{4t}{T}\right)^2} \quad 1$$

Equation 2 updates the position of the leader ( $x_j^1$ ) according to 1. The position of the source of food, 2. Lower and upper boundaries, 3. Three parameters ( $v_1$  which is calculated by Eq. 1, and  $v_2$ ,  $v_3$  that are randomly generated in the interval of (0,1))<sup>36-38</sup>.

$$x_j^1 = \begin{cases} S_j + v_1 ((u_j - l_j) v_2 + l_j) v_3 \geq 0 \\ S_j - v_1 ((u_j - l_j) v_2 + l_j) v_3 \geq 0 \end{cases} \quad 2$$

Equation 3 updates the position of the followers ( $x_j^i$ ) depending on the position of the previous salp, where  $i \geq 2$ <sup>36-38</sup>.

$$x_j^i = \frac{x_j^i + x_j^{i-1}}{2} \quad 3$$

**Proposed Discrete SSA**

In this research, the SSA is an optimization metaheuristic swarm algorithm chosen by the proposed system to present a novel method for honeyword generation. This choice was based on the ability of the SSA to obtain excellent solutions, successfully enhance the random solutions, converge toward the optimum solutions, and balance exploration and exploitation.

This research made many changes to the SSA to suit our particular aim (honeyword generation). The most substantial of these changes is the transformation of the algorithm into one able to handle discrete values (alphabet characters) instead of continuous values (numbers). The changes made to the SSA in order to convert it into the proposed discrete salp swarm algorithm are:

1. The password's tokens function as salps.
2. The alphabet token is considered the most important token because it serves as the main target for the intruder when attempting to guess the password; these tokens will be processed through the

mechanism of the SSA to solve the problem. In contrast, the digits and special characters will be processed through simple random generators.

3. The root alphabet token of the real password will function as the resource food (best solution).
4. The generated alphabet tokens of honeywords will function as salps that move toward the root alphabet token to obtain the best solution.
5. The proposed algorithm will pursue the multi-swarm approach. Hence, it will have several leaders, each of whom will be followed by several followers; the leaders are the best salps of the population.
6. The leaders will consist of the best 10% of the population size. The remaining 90% will be the followers, which are equally distributed to the leaders. For instance, if the population size is 100, then (leaders=10 and followers=90); if ten swarms are formed, then each swarm will have one leader and nine followers.
7. The suggested algorithm proposes four special salp movements toward the source of food: insert, delete, translocation, and swap.
8. The leader will test the four proposed salp movements, and the best move will be committed to its followers.
9. The lower and upper boundaries of the SSA will not be used, because the alphabet tokens of the honeywords will be committed by the alphabet (a to z).
10. Equation 1 of the SSA will not be used because the parameters of  $v_1$ ,  $v_2$ , and  $v_3$  will be replaced by a step size  $s_z$  towards the goal; the step size will be  $(0.3 * \text{alphabet token size})$ .
11. Equation 2 (leaders' movements) will be replaced by the proposed salp movements.
12. Equation 3 (followers' movements) will be replaced by the proposed salp movements, e.g., the follower will be committed by the movement of its leader.
13. The proposed algorithm, at the end of every iteration, will replace the worst salps with ones that are generated randomly.

The suggested honeyword approach was implemented for the legacy-UI, which is more user-friendly because it requires users to input only their username and password. The password should contain alphabet letters, digits, and special characters for the system to consider. The suggested method uses 25 sweetwords, which implies that if

$k=25$ , the intruder has a  $(1/25=4\%)$  chance of successfully picking the sugarword and a  $(1-4\%=96\%)$  chance of picking a honeyword. According to the suggested approach, the intruder will not be able to choose the sugarword even if one of its tokens is known because each token of the sweetwords has been repeated five times. The attacker has a  $(1/5=20\%)$  chance of picking the sugarword in this situation.

The suggested SSA treats each of the password tokens with different techniques according to their type. Every token type has a different generator (i.e., an alphabet, digits, or special characters generator). For the alphabet tokens, the suggested algorithm develops proposed salp movements and proposed evaluation criteria.

#### Proposed Discrete SSA Token Generators

Three proposed token generators operate in the suggested algorithm. The alphabet generator is the most critical and complex, so it will use the SSA technique for alphabet token generation. The digits and special characters generators use a simplified random generation technique. The following are the proposed generators:

1. The proposed SSA alphabet token generator: Because the intruder's preferred method of guessing the true password is the alphabet token, it is the most essential portion of the honeyword. This generator is the most difficult to use since it relies on the SSA technique to solve problems. For this generator, the password tokens function as salps. The alphabet token from the sugarword is utilized as input for the generators; this is the root from which the honeyword alphabet tokens are generated. The generator makes five copies of the alphabet generator's top four tokens, then splits the 20 tokens into four groups (columns). Each group includes five similar tokens. The alphabet root should be duplicated five times. As a result, there will be 25 alphabet tokens in the proposed SSA.
2. The proposed SSA digit token generator: The root of this generator is the sugarword's digit token, and it relies on random generation. The generator creates four tokens with the same length as the root. The generator then makes five copies of each of the four produced digit tokens, splitting the 20 tokens into four groups (rows) of five tokens each. After adding five copies of the digit root, the proposed SSA will have 25 total digits tokens.
3. The proposed SSA special character token generator:

The root of this generator is the sugarword's digit token, and it relies on random generation. The generator creates four tokens with the same length as the root. The generator then makes five copies of each of the four produced digit tokens, splitting the 20 tokens into four groups (rows) of five tokens each. After adding five copies of the digit root, the proposed SSA will have 25 total special characters tokens.

Example 1: For the proposed discrete SSA using the parameters listed in the parameters section, if the sugarword is (5diamond^). The generated sweetwords by the proposed SSA will be as follows:

5diamond d^	5sigmoi d^	5diagona l^	5diagona l^	5diaphon e^
9diamond d'	9sigmoi d'	9diagona l'	9diagona l'	9diaphon e'
0diamond d\	0sigmoi d\	0diagona l\	0diagona l\	0diaphon e\
3diamond d*	3sigmoi d*	3diagona l*	3diagona l*	3diaphon e*
6diamond d<	6sigmoi d<	6diagona l<	6diagona l<	6diaphon e<

### Proposed Salp Movements

The suggested algorithm proposes movements for the salp that are specific to the alphabet token. The movement of the honeyword alphabet token toward the sugarword alphabet token will mimic the movement of a biological salp toward food. The token movement will be applied as a change of the characters in the alphabet tokens. The step size of this movement will be represented by the change amount in the characters of the alphabet token. The proposed step size  $sz$  is  $(0.3 * \text{alphabet token size})$ . The four proposed movements are listed below.

1. Insert: Select a particular character's positions on the token at random, then insert randomly-selected characters into the selected positions.
2. Delete: Select and delete characters from the token at random.
3. Translocation: Select a character's positions on the token at random, then exchange them with one another.
4. Swap: Select a character's positions on the token at random, then change them out for other randomly-selected characters.

Example 2: For the proposed discrete SSA that uses step size  $=0.3 * (\text{alphabet token length})$  during the salp movement (token generating), if the sugarword

alphabet token is (sun) then  $sz = 0.3 * (3) = 0.9$ , so 1 character will be changed. The generated tokens are (stun, un, nus, son) in sequence.

### Proposed Evaluation Criteria

The proposed evaluation process will involve the generated alphabet tokens only and will be evaluated based on the root alphabet token of the sugarword. The suggested SSA includes proposed evaluation criteria for the generated alphabet tokens, which together form what is called the approximation factor. The approximation factor is determined as the sum of the four criterion values, which fall within a range of (0,1). As noted in the parameters section below, each criterion has a distinct value. The four proposed distinct SSA criteria are.

1. Character similarity: The degree of similarity between the characters of the root token and the characters of the generated token.
2. Length similarity: The comparability in length between the characters in the root token and the characters in the generated token.
3. Part of Speech (PoS) similarity: In terms of PoS, the root token and the generated token are identical.
4. Meaningful word: The generated token is a meaningful English word.

Example 3: For the proposed discrete SSA that uses parameters listed in (parameters section), if the sugarword alphabet token is (sun), the generated honeyword tokens with their approximation factor will be as follows (tun=0.933, suv=0.933, =0.933, sum=0.933, son= 0.933).

### The Proposed Discrete SSA Steps

The suggested system uses a proposed discrete SSA to generate the honeywords as a token generating process, in which the sugarword is tokenized into three different tokens: alphabet, digits, and special characters, and each is handled in a different generator (alphabet, digits, and special characters generator) before collecting the honeywords with the sugarword to present the sweetwords. The password tokens will be handled like salps, but with numerous changes, as described in the proposed discrete SSA section. As illustrated in Algorithm 2, the suggested algorithm's general steps consist of six primary parts. It's worth noting that the production steps (2,3, and 4) run in parallel.

**Algorithm 2. The general steps of the proposed discrete salp swarm algorithm.**

- Step one: Tokenization. The sugarword is parsed into three tokens according to its type: alphabet, numbers, and special characters token. Each token type will be treated in a distinct way using a different generator. The token will stay with no change if it appears in the username. It is just showing without changes in all the sweetwords.
- Step two: Alphabet generator. The alphabet tokens from step one will be sent to the generator sub-steps below by the proposed SSA:
- a: Set the parameters of the alphabet generator. Pop-size  $n$  (population of salp), max generation, source of food  $s$ = alphabet token received from step1, number of salp movements  $sm$ , step size  $sz$ , evaluation criteria  $ec$ , best salps size  $bs$ , number of leaders  $nl$ , and number of followers  $nf$ .
  - b: Generate the initial salps (alphabet tokens) population with  $n$  randomly.
  - c: Compute the fitness (approximation factor) of the salp population by considering evaluation criteria and the source of food  $s$ .
  - d: Population classification. Dividing the population into multiple swarms, every swarm has a leader and many followers. The leaders are the best salps of the population. The number of leaders  $nl$  will be equal to  $(nl=0.1*n)$ , and the number of followers  $nf$  ( $nf=0.9*n$ ). The followers will be distributed equally among the leaders.
  - e: For every salp (alphabet token) in the population:
    1. Every leader in the population makes the  $sm$  movements with consideration to the  $sz$ , and adopts the best move with consideration to the evaluation criteria.
    2. Every follower in the population makes the move that is adopted by their leader by considering the  $sz$  and the evaluation criteria. If the move is better than the current position then do it, else do nothing.
  - f: Compute the fitness (approximation factor) of the new salp population while considering evaluation criteria.
  - g: Replace the worst salps of the population with ones that are generated randomly.
  - h: Repeat sub-steps c to g until max generation.
  - i: Return the best salps with considering to  $bs$ , as the alphabet honeyword tokens.
- Step three: Digits generator. Examine the digits token received from step one to determine if it's on the list of years or the list of consecutive and frequented numbers. If the token appears in one of the lists, the system chooses tokens from the matching list at random. If not, the digits token is given to the digit's generator, which goes through the following sub-steps:
- a: Set the number of the generated digits tokens  $d$ .
  - b: Generate tokens considering  $d$ , by randomly choosing digits with the same length as the root token.
  - c: Return the  $d$  tokens as the digits of the honeyword tokens.
- Step four: Special characters generator. Following the receipt of the special characters token from step one, the token is given to the special characters generator, which has the sub-steps below:
- a: Set the number of the generated special characters tokens  $c$ .
  - b: Generate tokens considering  $c$ , by randomly choosing special characters of the same length as the root token.
  - c: Return the  $c$  tokens as the special characters honeyword tokens.
- Step five: Collect honeywords. Gather the honeywords tokens from the previous three steps, each with five copies.
- Step six: Return sweetwords. To present sweetwords, combine the sugarword tokens (each with five copies) with the honeyword tokens from the previous step. The locations of sweetwords are permuted at random. The total result of sweetwords is 25.

**The Proposed Discrete SSA Pseudocode**

Following up on Algorithm 2 of the proposed discrete SSA and the description of its three token

generators, the proposed discrete SSA pseudocode in Algorithm 3 is shown in this section.



**Algorithm 3. The pseudocode of the proposed discrete salp swarm algorithm.**

**Parameter**

Pop-size  $n$  (population of salp), max generation  $mg$ , source of food  $s$ = alphabet root token, number of salp movements  $sm$ , step size  $sz$ , evaluation criteria  $ec$ , best salps size  $bs$ , number of leaders  $nl$ , and number of followers  $nf$ , number of the generated digits tokens  $d$ , number of digits that changed in the generated token  $dl$ , number of the generated special characters tokens  $c$ , and number of special characters that changed in the generated token  $cl$ .

**Begin**

```
Tokenization /* parse the sugarword to the alphabet, digits, and special characters token */
If the token is an alphabet
    Generate the initial salps population with  $n$  randomly
    Compute the fitness of the population by considering  $ec$  and  $s$ 
    for  $i=1$  to  $mg$ 
        Let the best salps  $nl$  as leaders and the rest of the salps  $nf$  as followers, distributed over all leaders equally
        for  $j=1$  to  $n$ 
            if the salp is a leader
                makes the  $sm$  movements with considering to the  $sz$ , and adopt the best move with considering to  $ec$  and  $s$ 
            else makes the move that is adopted by its leader by considering the  $sz$ ,  $ec$ , and  $s$ 
                if the move is worse than the previous position then cancel it
            end if
        end for
        Compute the fitness of the new population by considering  $ec$  and  $s$ 
        Replace the worst salps of the population with ones that generate them randomly
    end for
Return the best salps with consideration to  $bs$ , as the alphabet honeyword tokens
end if
If the token is digits
    for  $i=1$  to  $d$ 
        for  $j=1$  to  $dl$ 
            Changes the digits of the token by other digits randomly
        end for
    end for
Return the  $d$  tokens as the digits honeyword tokens.
end if
If the token is special characters
    for  $i=1$  to  $c$ 
        for  $j=1$  to  $cl$ 
            Changes the special characters of the token by other special characters randomly
        end for
    end for
Return the  $c$  tokens as the special characters honeyword tokens.
end if
Collect honeyword tokens with five copies
Return sweetwords by adding five copies of sugarword tokens to honeywords tokens then permutate and hashed the sweetwords
```

**End**

**Parameters**

The proposed discrete SSA's performance is influenced by many of the parameters employed in the suggested honeyword generation method. The suggested algorithm was tested using a range of parameter values before those that provided the best performance for the suggested system were chosen.

The following are the proposed parameters, tested using multiple values.

- Population size (population of salps)  $n$ : The proposed SSA experimented with many population sizes (20, 40, 60, 80), with the size (80) being chosen as the generation size because the number of individuals suited the algorithm procedures.



- Max generation  $mg$ : Although several iterations were employed (10, 20, 30, 40..., 100), no improvement in outcomes was observed after 30 rounds. As a consequence, the alphabet token was assigned the maximum round's number possible (30).
- Step size  $sz$ : The change in token during salp movements was tested in many sizes, including 1 character, 2 characters,  $0.25*(\text{token length})$ ,  $0.3*(\text{token length})$ , and  $0.5*(\text{token length})$ ; the changing size ( $0.3*(\text{token length})$ ) was chosen because it provides words close to the original.
- Evaluation criteria  $ec$ : For the evaluation criteria (character similarity, length similarity, PoS similarity, and meaningful word), multiple value sets were selected for experimentation, including (0.3, 0.2, 0.2, 0.3), (0.4, 0.1, 0.1, 0.4), (0.3, 0.2, 0.1, 0.4), (0.3, 0.1, 0.1, 0.5), (0.2, 0.2, 0.1, 0.5), (0.2, 0.1, 0.2, 0.5), and (0.2,

0.2, 0.1, 0.5). Ultimately, the values (0.2, 0.1, 0.1, 0.6) were chosen, since they produce meaningful words. Such meaningful words work to confuse the intruder while the intruder tries to guess the real password.

Many parameters are used in the suggested honeyword generation approach that impacts the performance of the proposed discrete SSA. The parameters chosen to be used in the proposed discrete SSA are listed in Table 1.

**Table 1. The proposed discrete SSA parameters' values.**

No	Parameter	Value
1	Population size (population of salp) $n$	80
2	Max generation $mg$	30
3	Number of salp movement $sm$	4
3	Step size $sz$	$0.3*(\text{token length})$
4	Best salps size (number of the generated alphabet tokens) $bs$	4
5	Number of leaders salps $nl$	$0.1*(\text{population size})$
6	Number of followers salps $nf$	$0.9*(\text{population size})$
7	<u>Evaluation criteria <math>ec</math></u> Character similarity Length similarity PoS (part of speech) similarity Meaningful word	$\begin{pmatrix} 0.2 \\ 0.1 \\ 0.1 \\ 0.6 \end{pmatrix}$
8	Number of the generated digits tokens $d$	4
9	Number of digits that changed in the generated token $dl$	Token length
10	Number of the generated special characters tokens $c$	4
11	Number of special characters that changed in the generated token $sl$	Token length

### Experimental Results

The proposed SSA tested a range of password tokens, including the alphabet token, the most important token because the intruder's primary objective is to guess the real password. The experimental results are shown in Table 2 using the parameters indicated in Table 1 in the parameters section. The SSA strategy to solve the problem will

be used to generate the alphabet tokens. Eighty tokens will be constructed, but only the best four will be shown in Table 2. For the digits and special characters tokens, simple random generators will be utilized, with character changes occurring at random with the same root token length. Four tokens will be generated. Please see Example 1 for a complete example.

**Table 2. Experimental results of the proposed discrete SSA.**

Root Token	Pop-size/ Max-gen.	Honeyword Tokens/Approximation Factor					
1	<b>hello</b>	80/30	jello/0.96	hollo/0.96	cello/0.96	helve/0.919	
			rollo/0.919	mellon/0.916	hill/0.9	help/0.9	
			held/0.9	dell/0.9	hemlock/0.885	tulle/0.88	
			hel/0.88	hells/0.86	heir/0.86		
2	<b>smokey</b>	80/30	smiley/0.933	smote/0.916	smarmy/0.9	monkey/0.9	
			spoke/0.916	smote/0.916	turkey/0.9	spooky/0.9	
			smiley/0.933	smote/0.916	smoky/0.916	survey/0.9	
3	<b>batman</b>	20/30	matman/0.966	pitman/0.933	layman/0.933	banzai/0.9	
			40/30	bagman/0.966	potman/0.933	pitman/0.933	gasman/0.933
			60/30	barman/0.966	bagman/0.966	bitmap/0.933	banyan/0.933
			80/30	bataan/0.966	barman/0.966	bagman/0.966	layman/0.933
4	<b>teNNis</b>	80/30	teNias/0.933	teNpin/0.9	terNion/0.871	tiNier /0.866	
5	<b>tiger</b>	80/30	tiler/0.96	liger/0.96	taxer/0.919	taper/0.919	
6	<b>orange</b>	80/30	grange/0.966	fringe/0.933	flange/0.933	cringe/0.933	
7	<b>computer</b>	80/30	commuter/0.9	confuter/0.95	compete/0.937	accouter/ 0.9	
8	<b>purple</b>	80/30	supple/0.933	rumple/0.933	puzzle/0.933	pursue/0.933	
9	<b>scooter</b>	80/30	shooter/0.971	scepter/0.942	sootier/0.914	snifter/0.914	
10	<b>freedom</b>	80/30	freesia/0.914	freed/0.914	freebee/0.914	firedog/0.914	
11	<b>ofsye</b>	80/30	offset/0.883	okey/0.86	iffy/0.86	oxygen/0.85	
12	<b>pklser</b>	80/30	polder/0.933	palmer/0.933	falser/0.933	pulse/0.916	
13	<b>nusi</b>	80/30	monody/0.9	minder/0.9	manner/0.9	manda/0.85	
14	<b>9725</b>	N/A	7845	0715	5814	8068	
15	<b>631</b>	N/A	449	500	973	047	
16	<b>52</b>	N/A	59	14	74	09	
17	(% !-	N/A	[!~\$	#@+{	\$^!	} :_	
18	%.#	N/A	!\$*	=&.	?(!	&={	
19	~*	N/A	^!	](	..	@\	

Table 2 shows the proposed SSA-generated tokens for different types of tokens to declare the ability to handle every password token type. Token 1 (hello) shows that the proposed SSA can generate many good tokens with an approximation factor over the 0.6 value. Of those, 15 tokens exceeded the 0.6 value. Token 2 (smokey) shows that the proposed SSA generates different tokens for every attempt, even using the same tokens and Pop-size/Max-gen. Token 3 (batman) shows generated tokens in different Pop-size/Max-gen. There are always good results, but it is best when Pop-size=80 and Max-gen=30. Token 4 (teNNis) shows the ability of the proposed SSA to handle the capital letters of the password. Tokens 5-10 show different meaningful words as alphabet tokens. Tokens 11-13 show the generated tokens for meaningless words as alphabet tokens. Tokens 14-16 show digits' tokens. Tokens 17-18 show special characters' tokens.

### Comparison:

This section of the paper will compare the honeyword system proposed in this study to the honeyword generation methods described in previous literature. In terms of honeyword generation, the suggested SSA honeyword generation strategy outperforms previous techniques

by improving the generation process through its use of problem-solving features (i.e., obtaining the best solutions, efficiently improving random solutions, converging toward the best solution, and balanced exploration and exploitation).

The suggested SSA improves the most significant honeyword features (flatness, DoS Resistance, and storage), which previous honeyword generation methods did not effectively address. 1.) Flatness: The suggested SSA unconditionally ensures perfect flatness, with the intruder having a (1/25=4%) probability of successfully picking the sugarword and a (1-4% = 96%) chance of selecting a honeyword. The proposed SSA ensures that the intruder has a (1/5=20%) chance of picking the sugarword even if the intruder knows one of its tokens. 2.) DoS Resistance: A DoS attack works by guessing and inputting a honeyword to deny the system's services. The suggested SSA generates honeywords that are impossible for intruders to predict, while it also saves usernames and sweetwords. 3.) Storage: Some previous generation approaches store more data and information that exceeds the honeyword storing capacity, while the proposed system does not.

The following provides a specific comparison between previous honeyword generation methods and the suggested honeyword system, illustrating the seven most critical issues encountered by honeyword systems. Furthermore, a table will encompass the generation methods and show which method experiences the issue and which does not. Experiencing each described issue presents a weakness in the system while avoiding each issue presents a system strength. The suggested honeyword system succeeded in overcoming the seven issues common to previous honeyword systems. The seven issues are as follows.

1. Conditionally flatness issue: The fulfillment of some conditions to achieve perfect flatness is considered a weakness, whereas unconditional flatness means that no condition considered a strength must be met. Most previous honeyword generation systems only ensure perfect flatness under specific conditions; in contrast, the proposed honeyword system provides perfect flatness unconditionally.
2. Weak DoS Resistance issue: When weak DoS resistance is in place, an intruder can easily guess the system's honeywords. On the other hand, good DoS resistance ensures that the intruder will be unable to guess them. While many previous honeyword generations' approaches included weak DoS resistance, the proposed honeyword system has strong DoS resistance.
3. Storage overhead issue: More storage space has often been required to effectively operate previous honeyword generation systems, storage that exceeds the honeywords storage. Unlike many such previous systems, the proposed honeyword system does not require additional storage expenditures.
4. Correlation issue: The existence of a correlation between username and password is problematic. Such a correlation increases the possibility that an intruder can distinguish the honeywords from the true password. By maintaining the correlated component within the honeywords, the proposed honeyword system overcomes that issue.
5. Consecutive and frequented numbers issue: Users tend to gravitate toward numerical patterns that are easy to recall. As a result, many people choose to utilize numbers in their passwords that are consecutive or frequented, such as "123," "1234," "111," or "2222." While convenient for users, this tendency leaves the sugarword vulnerable to the identification. To address this issue, the proposed honeyword system suggests a list with the most frequented and consecutive numbers. If the sugarword includes numbers on this list, the proposed algorithm will choose numbers at random from the list for the honeywords.
6. Special date issue: Many users like to include a date in their passwords that is significant to them, such as a birthday, anniversary, graduation year, or any other date that may facilitate recall but also exposes the sugarword to discovery. Therefore, in the proposed honeyword system, a list of the previous 50 years will be generated. If a year number from the list is used in the sugarword, the system will choose years randomly from the list for the honeywords.
7. User's information, security issue: Many previous honeyword generation methods have relied on personal information questions, which require the user to provide personal information for the system to work. If the system is hacked, personal information may be revealed and used on another system, endangering the user. Thus, using this strategy with its inherent security risks is considered a weakness, whereas not using it is a strength. As a result, the proposed honeyword system does not ask the user to provide any personal information; it just asks him to provide a username and password.

In the most important honeyword system issues, Table 3 compares the proposed SSA with past honeyword generating systems.

**Table 3. A comparison in the most critical issues of honeyword systems.**

No	Methods	Cond. Flatness issue	Weak DoS resist. issue	Storage overhead issue	Correlation issue	Cons. And frequent numbers issue	Special Date issue	User Info. security issue
1	Proposed SSA	No	No	No	No	No	No	No
2	Chaffing-by-tail-tweaking <sup>16</sup>	Yes	Yes	No	Yes	Yes	Yes	No
3	Chaffing-by –tweaking-digits <sup>16</sup>	Yes	Yes	No	Yes	Yes	Yes	No
4	Simple model <sup>16</sup>	Yes	No	No	Yes	No	No	No
5	Modeling syntax <sup>16</sup>	Yes	No	No	Yes	Yes	Yes	No
6	Chaffing with “tough nuts” <sup>16</sup>	N/A	No	Yes	No	N/A	N/A	No
7	Take-a-tail <sup>16</sup>	No	No	No	No	No	No	No
8	Random pick <sup>16</sup>	Yes	No	No	Yes	No	No	No
9	Hybrid generation methods <sup>16</sup>	Yes	No	No	Yes	Yes	Yes	No
10	Storage-index <sup>17</sup>	Yes	Yes	Yes	Yes	No	No	No
11	PDP <sup>18</sup>	Yes	No	Yes	No	Yes	No	No
12	Evolving password model <sup>19</sup>	Yes	No	No	Yes	Yes	Yes	No
13	User-profile model <sup>19</sup>	Yes	Yes	Yes	Yes	Yes	No	Yes
14	Append-secret model <sup>19</sup>	Yes	No	No	No	Yes	No	No
15	User information method <sup>20</sup>	Yes	Yes	Yes	Yes	Yes	No	Yes
16	Dictionary attack method <sup>20</sup>	Yes	Yes	No	Yes	No	No	No
17	Generic password list method <sup>20</sup>	Yes	No	No	Yes	No	No	No
18	Shuffling characters method <sup>20</sup>	Yes	Yes	No	Yes	Yes	Yes	No

### Discussion:

The experimental results revealed that the proposed technique creates passwords using all of its tokens (alphabet, digits, and special characters, but most notably the alphabet token) that are particularly tough due to their ability to mimic meaningful sentences. The generation of alphabet tokens produced effective results in terms of generating meaningful words out of meaningful words; more significantly, however, the system was also able to generate meaningful words out of meaningless words. The suggested SSA concludes that the population size (pop-size) should be greater than the maximum generation (max-gen) as a result of the outcomes analysis; the proposed system thus selects pop-size=80/max-gen=30 based on experience. While pop-sizes of (20, 40, 60, 80) offer good results, Pop-size=80 produces a superior approximation factor according to the experimental results. Based on these results, the honeywords created include several beneficial qualities: each password token type is generated independently, even though the pop-size/max-gen is constant for every generating operation; the generated tokens differ from one process to another; the system can conduct a variety of token order password patterns; the process offers great protection against intruder guessing, and the system can handle the capital letters of alphabet tokens.

The comparisons described in the previous section between the proposed SSA and earlier generation strategies show that the

suggested method is superior in three major areas: honeyword production, honeyword features, and addressing the prominent issues of previous methods, as shown in the testing results. The system’s most important feature, flatness, demonstrates a considerable improvement over previous approaches—the proposed system exhibits a superior flatness ( $1/25=4\%$ ). Furthermore, even if an intruder knows one of the sugarword tokens, the intruder still has only a ( $1/5=20\%$ ) chance of selecting the sugarword.

### Conclusion

Salp swarm algorithm (SSA) is an optimization metaheuristic swarm bio-inspired algorithm employed in this research to provide a novel technique for the honeyword production process. The proposed SSA has been subjected to numerous changes in order to address the problem space; as a result, the proposed SSA generates honeywords as solutions. The proposed system efficiently uses an intelligence algorithm (SSA) for security purposes, specifically password cracking detection (honeyword system). The proposed system creates honeywords by utilizing SSA’s solution generation properties (obtaining excellent solutions, efficiently improving random solutions, converging toward the optimal solution, and balancing exploration and exploitation). The proposed discrete SSA improves the generation process, improves honeyword properties, and overcomes the drawbacks of previous techniques.

The alphabet token is the sugarword's most important and difficult token. Therefore, the proposed system creates the alphabet token in the solution of the problem using the proposed SSA approach. The digit and special characters tokens, on the other hand, rely on a simpler random generation process.

It is important to note that, if the initial population is not well-diversified, a limitation might be introduced into the suggested approach, resulting in additional execution iterations.

Based on the information gathered from this investigation into the employment of metaheuristic algorithms, this work proposes honeyword-producing strategies and aims to develop another intelligence methodology that may provide perfect solutions. Future researchers can apply the proposed SSA to their own research to determine how the proposed system can be utilized to solve multi-objective optimization problems. Future research into this area might also lead to the identification and resolution of other issues commonly confronting honeyword systems. Certain aspects of the SSA could further be enhanced and hybridized with another method.

#### Acknowledgment:

Appreciate the cooperation of University of Technology, Baghdad, Iraq, and University of the Cumberland, KY, USA.

#### Authors' Declaration:

- Conflicts of Interest: None.
- We hereby confirm that all the Algorithms and Tables in the manuscript are ours. Besides, an algorithm, which is not ours, has been permitted re-publication.
- Ethical Clearance: The project was approved by the local ethical committee at University of Technology, Iraq, Baghdad.

#### Authors' contributions statement:

Y. A. Y. is contributed to the article by the conception, design, analysis, interpretation, and drafting of the MS manuscript. A. T. S. shared in the conception, analysis, interpretation, revision, and proofreading. W. Al. participated in the conception, revision, and proofreading.

#### References:

1. Mohammed AA, Abdul-Hassan AK, Mahdi BS. Authentication System Based on Hand Writing Recognition. In: 2019 SCCS 2019 - 2019 2nd Sci. Conf. Comput. Sci; 2019: 138-142. doi:10.1109/SCCS.2019.8852594
2. Abdulameer SA, Kashmar AH, Shihab AI. A cryptosystem for database security based on TSFS algorithm. Baghdad Sci J. 2020; 17(2): 567-574. doi:10.21123/bsj.2020.17.2.0567
3. Alaa Kadhim F, Mhaibes HI. A New Initial Authentication Scheme for Kerberos 5 Based on Biometric Data and Virtual Password. ICOASE 2018 - Int Conf Adv Sci Eng. Published online 2018: 280-285. doi:10.1109/ICOASE.2018.8548852
4. Genç ZA, Kardaş S, Kiraz MS. Examination of a New Defense Mechanism: Honeywords. In: Hancke GP, Damiani E, eds. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 10741. Lecture Notes in Computer Science. Springer International Publishing; 2018: 130-139. doi:10.1007/978-3-319-93524-9\_8
5. Win T, Moe KSM. Protecting private data using improved honey encryption and honeywords generation algorithm. Adv Sci Technol Eng Syst. 2018; 3(5): 311-320. doi:10.25046/aj030537
6. Chakraborty N, Mondal S. Towards Improving Storage Cost and Security Features of Honeyword Based Approaches. Procedia Comput Sci. 2016; 93(September):799-807. doi:10.1016/j.procs.2016.07.298
7. Palaniappan S, Parthipan V, Stewart kirubakaran S, Johnson R. Secure User Authentication Using Honeywords. In: Lecture Notes on Data Engineering and Communications Technologies. 31. 2020: 896-903. doi:10.1007/978-3-030-24643-3\_105
8. Wang R, Chen H, Sun J. Phoney: Protecting password hashes with threshold cryptology and honeywords. Int J Embed Syst. 2016; 8(2-3): 146-154. doi:10.1504/IJES.2016.076108
9. Homayouni SM, Fontes DBMM. Metaheuristic Algorithms. In: Metaheuristics for Maritime Operations. Volume 1. John Wiley & Sons, Inc. 2018; ch2: 21-38. doi:10.1002/9781119483151.ch2
10. Tezel BT, Mert A. A cooperative system for metaheuristic algorithms. Expert Syst Appl. 2021; 165(May 2020): 113976. doi:10.1016/j.eswa.2020.113976
11. Malik H, Iqbal A, Joshi P, Agrawal S, Farhad IB. Metaheuristic and Evolutionary Computation: Algorithms and Applications. Part of the Studies in Computational Intelligence book series. Springer Singapore. 916. 2021. doi:10.1007/978-981-15-7571-6
12. Yasear SA, Ku-Mahamud KR. Taxonomy of memory usage in swarm intelligence-based metaheuristics. Baghdad Sci J. 2019; 16(2): 445-452. doi:10.21123/bsj.2019.16.2(SI)0445
13. Faeq IF, Duaimi MG, Sadiq Al-Obaidi AT. An efficient artificial fish swarm algorithm with harmony search for scheduling in flexible job-shop problem. J Theor Appl Inf Technol. 2018; 96(8): 2287-2297. <http://www.jatit.org/volumes/Vol96No8/18Vol96No8.pdf>
14. Castelli M, Manzoni L, Mariot L, Nobile MS, Tangherloni A. Salp Swarm Optimization: A critical review. Expert Syst Appl. 2021 (November): 116029. doi:10.1016/j.eswa.2021.116029
15. Faris H, Mirjalili S, Aljarah I, Mafarja M, Heidari

- AA. Salp swarm algorithm: Theory, literature review, and application in extreme learning machines. *Stud Comput Intell.* 2020; 811(January): 185-199. doi:10.1007/978-3-030-12127-3\_11
16. Juels A, Rivest RL. Honeywords: Making Password-Cracking Detectable. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13.* ACM Press, 2013: 145-160. doi:10.1145/2508859.2516671
17. Erguler I. Achieving Flatness: Selecting the Honeywords from Existing User Passwords. *IEEE Trans Dependable Secur Comput.* 2015; 13(2): 284-295. doi:10.1109/TDSC.2015.2406707
18. Chakraborty N, Mondal S. On designing a modified-UI based honeyword generation approach for overcoming the existing limitations. *Comput Secur.* 2017; 66: 155-168. doi:10.1016/j.cose.2017.01.011
19. Akshima A, Chang D, Goel A, Mishra S, Sanadhya SK. Generation of Secure and Reliable Honeywords, Preventing False Detection. *IEEE Trans Dependable Secur Comput.* 2018; 5971(c): 1-13. doi:10.1109/TDSC.2018.2824323
20. Akif OZ, Sabeeh AF, Rodgers GJ, Al-Raweshidy HS. Achieving flatness: Honeywords generation method for passwords based on user behaviours. *Int J Adv Comput Sci Appl.* 2019; 10(3): 28-37. doi:10.14569/IJACSA.2019.0100305
21. Lanjulkar Pritee, Ingle Rupali, Lonkar Arti IV. Honeywords: A New Approach For Enhancing Security. *Int Res J Eng Technol.* 2019; 06(03): 1360-1363. <https://www.irjet.net/archives/V6/i3/IRJET-V6I3256.pdf>
22. Veera Babu R, Praneerhasrurhi M. Security Enhancement by Achieving Flatness in Selecting the Honey words from Existing User Passwords. *Int J Eng Tech.* 2018; 4(2): 743-746. <http://www.ijetjournal.org/Volume4/Issue2/IJET-V4I2P115.pdf>
23. Weiwei Jing, Jinku Cui YZ. A Honeyword Generation Method Based on Special Character Distance. *Softw Eng Appl.* 2019; 08(05): 207-214. doi:10.12677/SEA.2019.85025
24. Ghare H. Securing System using Honeyword and MAC Address. *Int J Res Appl Sci Eng Technol.* 2019; 7(5): 2685-2689. doi:10.22214/ijraset.2019.5446
25. Thakur PV. Honeywords: The New Approach for Password Security. *Int J Res Appl Sci Eng Technol.* 2019;7(4):2449-2450. doi:10.22214/ijraset.2019.4446
26. Shinde PD, Patil SH. Secured Password Using Honeyword Encryption. *Iioab J.* 2018; 9(2): 78-82. [https://www.iioab.org/IIOABJ\\_9.2\\_78-82.pdf](https://www.iioab.org/IIOABJ_9.2_78-82.pdf)
27. Guo Y, Zhang Z, Guo Y. Superword: A honeyword system for achieving higher security goals. *Comput Secur.* 2021; 103: 101689. doi:10.1016/j.cose.2019.101689
28. Bamane S. Achieving Flatness Using Honeywords Generation Algorithm. *Int J Res Appl Sci Eng Technol.* 2019; 7(5): 3491-3496. doi:10.22214/ijraset.2019.5572
29. Kute S, Thite V, Chopade S. Achieving Security using Honeyword. *Int J Comput Appl.* 2018; 180(49): 43-47. doi:10.5120/ijca2018917333.
30. Çelik E, Öztürk N, Arya Y. Advancement of the search process of salp swarm algorithm for global optimization problems. *Expert Syst Appl.* 2021; 182(March). doi:10.1016/j.eswa.2021.115292
31. Bairathi D, Gopalani D. An improved salp swarm algorithm for complex multi-modal problems. *Soft Comput.* 2021; 25(15): 10441-10465. doi:10.1007/s00500-021-05757-7
32. Gupta S, Deep K, Heidari AA, Moayedi H, Chen H. Harmonized Salp Chain-Built Optimization. 37. Springer London; 2021. doi:10.1007/s00366-019-00871-5
33. Balakrishnan K, Dhanalakshmi R, Khaire UM. Improved salp swarm algorithm based on the levy flight for feature selection. *J Supercomput.* 2021; 77(11): 12399-12419. doi:10.1007/s11227-021-03773-w
34. Hegazy AE, Makhlof MA, El-Tawel GS. Improved salp swarm algorithm for feature selection. *J King Saud Univ - Comput Inf Sci.* 2020; 32(3): 335-344. doi:10.1016/j.jksuci.2018.06.003
35. Ouair F, Boudjemaa R. Modified salp swarm algorithm for global optimisation. *Neural Comput Appl.* 2021 (July). doi:10.1007/s00521-020-05621-z
36. Mirjalili S, Gandomi AH, Mirjalili SZ, Saremi S, Faris H, Mirjalili SM. Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. *Adv Eng Softw.* 2017; 114: 163-191. doi:10.1016/j.advengsoft.2017.07.002
37. Abualigah L, Shehab M, Alshinwan M, Alabool H. Salp swarm algorithm: a comprehensive survey. *Neural Comput Appl.* 2020; 32(15): 11195-11215. doi:10.1007/s00521-019-04629-4
38. Slowik A. *Swarm Intelligence Algorithms.* CRC Press, 1st Ed. 2020. 362P. doi:10.1201/9780429422614

## توليد الكلمات العسلية باستخدام خوارزمية سرب عنب البحر المتقطعة

وسيم الحمداني<sup>2</sup>احمد طارق صادق<sup>1</sup>ياسر علي ياسر<sup>1</sup><sup>1</sup> قسم علوم الحاسوب، الجامعة التكنولوجية، بغداد، العراق.<sup>2</sup> قسم تكنولوجيا المعلومات، جامعة كمبرلاند، كنتاكي، الولايات المتحدة الأمريكية.

## الخلاصة:

إن كلمات العسل (Honeywords) هي كلمات مرور مزيفة مرافقة لكلمة المرور الحقيقية والتي تدعى كلمة السكر. يعد نظام كلمات مرور العسل نظامًا فعالاً لاكتشاف اختراق كلمات المرور مصمم لاكتشاف اختراق كلمة المرور بسهولة من أجل تحسين أمن كلمات المرور المشفرة. لكل مستخدم ، سيكون لملف كلمة المرور الخاص بنظام الكلمات العسلية كلمة مرور واحدة حقيقية مشفرة مصحوبة بالعديد من كلمات المرور المزيفة المشفرة. إذا قام شخص دخيل بسرقة ملف كلمات المرور من النظام ونجح في اختراق كلمات المرور محاولاً تسجيل الدخول إلى حسابات المستخدمين ، فسيكتشف نظام كلمات المرور هذه المحاولة من خلال مدقق العسل (Honeychecker) مدقق العسل هو خادمًا إضافيًا يميز كلمة المرور الحقيقية عن كلمات المرور المزيفة ويطلق إنذارًا إذا قام شخص دخيل بتسجيل الدخول باستخدام كلمة مرور العسل. تم اقتراح العديد من طرق توليد كلمات العسل خلال البحوث السابقة، مع وجود قيود على عمليات إنشاء كلمات العسل الخاصة بهم ، ونجاح محدود في توفير جميع ميزات كلمات العسل المطلوبة ، والتعرض للعديد من مشكلات كلمات العسل. سيقدم هذا العمل طريقة جديدة لتوليد كلمات العسل تستخدم خوارزمية سرب عنب البحر المتقطعة. خوارزمية سرب عنب البحر هي خوارزمية تحسين مستوحاة من الأحياء تحاكي سلوك سرب عنب البحر في بيئتها الطبيعية. تم استخدام خوارزمية سرب عنب البحر لحل مجموعة متنوعة من مشاكل التحسين. ستعمل طريقة توليد الكلمات العسلية المقترحة على تحسين عملية توليد كلمات العسل وتحسين ميزات كلمات العسل والتغلب على عيوب التقنيات السابقة. ستوضح هذه الدراسة العديد من الاستراتيجيات السابقة لتوليد الكلمات العسلية، ووصف الطريقة المقترحة، وفحص النتائج التجريبية، ومقارنة طريقة إنتاج كلمات العسل الجديدة بالطرق السابقة.

**الكلمات المفتاحية:** المصادقة، كلمة المرور، كلمة العسل، خوارزمية سرب عنب البحر، خوارزمية السرب.