BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **22**, No 1
Sofia • 2022 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2022-0003

A Proposal for Honeyword Generation via Meerkat Clan Algorithm

Yasser A. Yasser[1]*, Ahmed T. Sadiq*[1]*, Wasim AlHamdani*[2]

[1]*Computer Science Department, University of Technology-Iraq, Iraq*
[2]*Information Technology Department, University of the Cumberlands, KY, USA*
E-mails: cs.19.28@grad.uotechnology.edu.iq Ahmed.T.Sadiq@uotechnology.edu.iq wasim.alhamdani@ucumberlands.edu

***Abstract*:** *An effective password cracking detection system is the honeyword system. The Honeyword method attempts to increase the security of hashed passwords by making password cracking easier to detect. Each user in the system has many honeywords in the password database. If the attacker logs in using a honeyword, a quiet alert trigger indicates that the password database has been hacked. Many honeyword generation methods have been proposed, they have a weakness in generating process, do not support all honeyword properties, and have many honeyword issues. This article proposes a novel method to generate honeyword using the meerkat clan intelligence algorithm, a metaheuristic swarm intelligence algorithm. The proposed generation methods will improve the honeyword generating process, enhance the honeyword properties, and solve the issues of previous methods. This work will show some previous generation methods, explain the proposed method, discuss the experimental results and compare the new one with the prior ones.*

***Keywords*:** *Honeyword, password, metaheuristic, swarm, meerkat.*

1. Introduction

Password-based authentication is the most frequently accepted authentication mechanism because of its ease of implementation and memorability [1]. However, this technique has been investigated using multiple attack types such as password cracking. Password cracking retrieves passwords from data saved or delivered through a computer system using an unusual and typically immoral approach [2].

Honeywords is a simple way for strengthening the security of hashed passwords and making password cracking easier to detect by increasing the amount of "honeywords" (False passwords) associated with each user's account [3, 4]. If an attacker obtains access to the file of hashed passwords and reverses the hash algorithm will not infer the true password. A "silent alarm" will be triggered if a honeyword is used during the login procedure [5, 6]. Honeychecker is an auxiliary server that can recognize the true password and the honeyword, it is connected to the login server using a secure connection [7, 8].

In computer science and mathematical optimization, a metaheuristic is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem [9]. An optimization issue is a problem in mathematics, computer science, and economics where the goal is to identify the optimum answer out of all possible ones [10]. The metaheuristic algorithms can be, swarm or nature-inspired. Swarm Intelligence (SI) Algorithms are the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence [11, 12]. Nature-inspired Algorithms represent a set of novel problem-solving methodologies on artificial intelligence [13]. One of the metaheuristic swarm intelligence algorithms is the Meerkat Clan Algorithm (MCA), which results from the watchful observation of the meerkat while looking for food aims to resolve optimization problems by determining the optimum solution [14-16].

Many honeyword generation methods have been proposed, they have a weakness in generating process, do not support all honeyword properties, and have many honeyword issues. This article proposes a novel method to generate honeyword using the meerkat clan intelligence algorithm, a metaheuristic swarm intelligence algorithm. The meerkat clan algorithm generates solutions, and the proposed method employs the algorithm to generate honeywords, benefiting from its properties in diversity and fast convergence to solutions. The generation methods being proposed will improve the honeyword generating process, enhance the honeyword properties, and solve the issues of previous methods. Furthermore, this study lists a few related works of honeyword generation methods, provide a simple explanation for honeyword technique, illustration for the meerkat algorithm, explanation for the proposed system with proposed MCA, presentation for the experimental results, and comparisons with the previous method.

2. Related works

There is a lot of research that has proposed the honeyword generation methods through the last years. This section contains some asymptotical studies.

- Juels and Rivest [17] propose a variety of honeyword generating methods that depend on: tweaking a part of the password, using a dictionary, appending a tail by the system, honeywords suppling by the system, honeywords suppling by the user, or hybrid methods. These methods divide into two categories according to whether or not they influence the User Interface (UI), each of the two categories has many techniques to generate honeyword:

1. Legacy-UI based honeyword generation methods:

Chaffing-by-tail-tweaking, Chaffing-by–tweaking-digits, Simple model, Modeling syntax, “Tough nuts”, Hybrid generation methods.

2. Modified-UI based honeyword generation methods:

Take-a-tail, Random pick.

• E r g u l a r [18] proposes a method called “Storage-index”, and it suggests an alternate way for honeyword creation that chooses honeywords using current user passwords in the system to generate realistic honeywords. Instead of creating honeywords and storing them in a password file, this method mimics honeywords using existing passwords.

• C h a k r a b o r t y and M o n d a l [19] propose Paired Distance Protocol (PDP), which is a new honeyword generation mechanism that uses a new user interface. Three portions of information are needed to log in: a username, a password, and a password-tail. User can pick his password-tail with this method. On registering, 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).

• A k s h i m a et al. [20] propose as better and more useful honeyword generation approach, the “evolving-password model”, “user-profile model”, and “append-secret model”.

a) Evolving-password model: Two independent calculation phases may be used to conduct the complete procedure, counting the number of times password patterns and tokens are used. Creating honeywords from pre-calculated frequencies and keeping frequency lists up to date.

b) User-profile model: Honeywords are created by combining various user profile information through the construction of separate sets from given user information that comprise tokens of each kind, such as “alphabet-strings”, “digit-strings”, and “special-character-strings”.

c) Append-secret model: The system asks the user for username, password, and an extra entry, say *e*, to generate a random string *s*, taking into consideration numbers, letters, and symbols. The model runs the function $f(p||e||s)$ and returns *r*. The system’s password file will save *H* (password$||r$).

• A k i f et al. [21] propose a new strategy for generating honeywords that incorporates four strategies. As a result, four sets of honeywords appear to be real passwords injected into the system.

a) Generate honeywords from existing user information: Creating a database with public personal questions divided into two parts – the first part will focus on characters and the second part will focus on numbers.

b) Generate honeywords from a dictionary attack: After skimming through the dictionary attack, the main principle behind constructing acceptable honeywords is to use the original password with a change of up to three digits or characters.

c) Generate honeywords from a generic password list: This honeyword group is made up of honeywords chosen at random from a list of the 500 worst passwords.

d) Generate honeyword form shuffling the characters: Honeyword is made by shuffled letters or digits from the ID user and then mixed in.

3. Honeywords

The honeywords system is based on the creation of honeywords (fake passwords) from the sugarword (true password) and then inserting them all into the user account as sweetwrods, then hashing them all [22, 23]. If the attacker is successfully obtained plain passwords from hashed passwords, he must then make a correct guess for the true password among the sweetwords; instead, a silent warning to the system administrator may be triggered, indicating that password cracking is possible [24, 25]. The action taken by the administrators is determined by the organization's policy and may include blocking, postponing, or alerting the account [26, 27].

4. Meerkat Clan intelligence Algorithm

The metaheuristics Meerkat Clan Algorithm (MCA) is a swarm nature-stimulated algorithm. The MCA is a global optimization metaheuristic, which is gathered by selecting the optimal structure and a randomization structure. The MCA's generated solutions have many properties (proximity principle, quality principle, diverse response principle, stability principle, and adaptability principle) [14]. Meerkats are sociable creatures that live in groups of several individuals, each group has its territory. If food cannot be obtained or if a stronger group demands it the weaker group will either try to grow in a different method or stay until they get stronger and reclaim their lost territory [15]. Each group also has a sentry, or someone who watches over the group and knows when to notice danger and alert the other individuals. The sentry keeps an eye on both the burrow plan and the other group members hunting for food [16]. As an objective function A l-O b a i d i, A b d u l l a h and A h m e d [14] use the Euclidean distances in solving the traveling salesman's problem TSP, while J a m e e l and A b d u l l a h [16] use the mean absolute deviation MAD in solving a feature selection problem.

Following up on the previous description of the MCA, Algorithm 1 showing the generic stages of this algorithm, which may be modified depending on the issue encoded [14].

Algorithm 1. The generic steps of the meerkat clan algorithm [14]

a. **Initialization:** generate a clan of people at random and establish the clan size, foraging size, care size, and worst foraging and care rate parameters.

b. Calculate the clan's fitness.

c. Select the best one as "sentry".

d. Split the clan into two parts (foraging and care).

e. Create neighbors for the foraging group.

f. Pick the worst members of the foraging group and replace them with the finest members of the care group.

g. Remove the worst members of the care group and generate a new one at random.

h. If the best individual in foraging is sentry, replace him.

5. Proposed system

This study proposes a new honeyword system that suggests a novel method for generating honeyword by using a metaheuristic swarm algorithm called the Meerkat Clan intelligence Algorithm (MCA). The proposed system uses three different algorithms for generating a process, the proposed MCA alphabet generator, the random digits generator, and the random special characters generator.

The choice for MCA to generate the alphabet honeyword tokens was because of utilization, examination, and its principles on find solutions (proximity principle, quality principle, diverse response principle, stability principle, and adaptability principle), note that the system is handling solutions as honeywords.

The legacy-UI is adopted in the system. When the user creates the account, all he has to do is provide the username and password. The password should contain alphabets, digits, and special characters.

The proposed system chooses the number of sweetwords=49, as the k=49, the attacker has a 1/49(≈2%) probability of picking a sugarword and 1 – 2%=98% chance of picking a honeyword in the perfectly flat honywords generation. The sugarword cannot be guessed even if the adversary knows one of the sugarword tokens. In sweetwords, each token is redundant seven times, if the adversary knows one of the sugarword tokens, then he has the chance of picking the sugarword at random by 1/7(≈14%).

The proposed system's most important aims are to improve the honeyword generating process, enhance the honeyword properties, and solve the issues of previous methods (detailed discussion in Section 7.2. Comparisons).

The system acts by six main steps to generate honeywords from the sugarword (tokenization, alphabet generating, digits generating, special characters generating, collect honeywords, and sweetwords), the generating steps work in parallel. Fig. 1 shows the suggested system's block-diagram.

- **Step 1 (Tokenization).** Parsing the sugarword characters into distinct tokens depends on their type – alphabet, digits, and special characters. If the token is found in the username, then the system lets the correlated part stay as is in the honeywords.
- **Step 2 (Alphabet generating).** Receive the alphabet tokens from step one, then send them to the proposed MCA alphabet generator.
- **Step 3 (Digits generating).** Receive the digits token from step one, then check if it is in the year's list or consecutive and frequented numbers list. If the token is in one of the lists, then the system randomly chooses tokens from the corresponding list to the honeywords. Else, the system sends the digits token to the random digits generator.
- **Step 4 (Special characters generating).** Receive the special characters token from step one, then send it to the random special characters generator.
- **Step 5 (Collect honeywords).** Collect the honeywords tokens from the three previous steps.
- **Step 6 (Sweetwords).** Add the sugarword to the honeywords to provide the sweetwords then, randomly permutate the positions of sweetwrods, the total number of sweetwords is 49 as presented in the next section. Send the index of the user $u(i)$ and the index of the sugarword $c(i)$ to the honeychecker. Then, hashing and store sweetwords. Note: Never store the index of sugarword in login server.

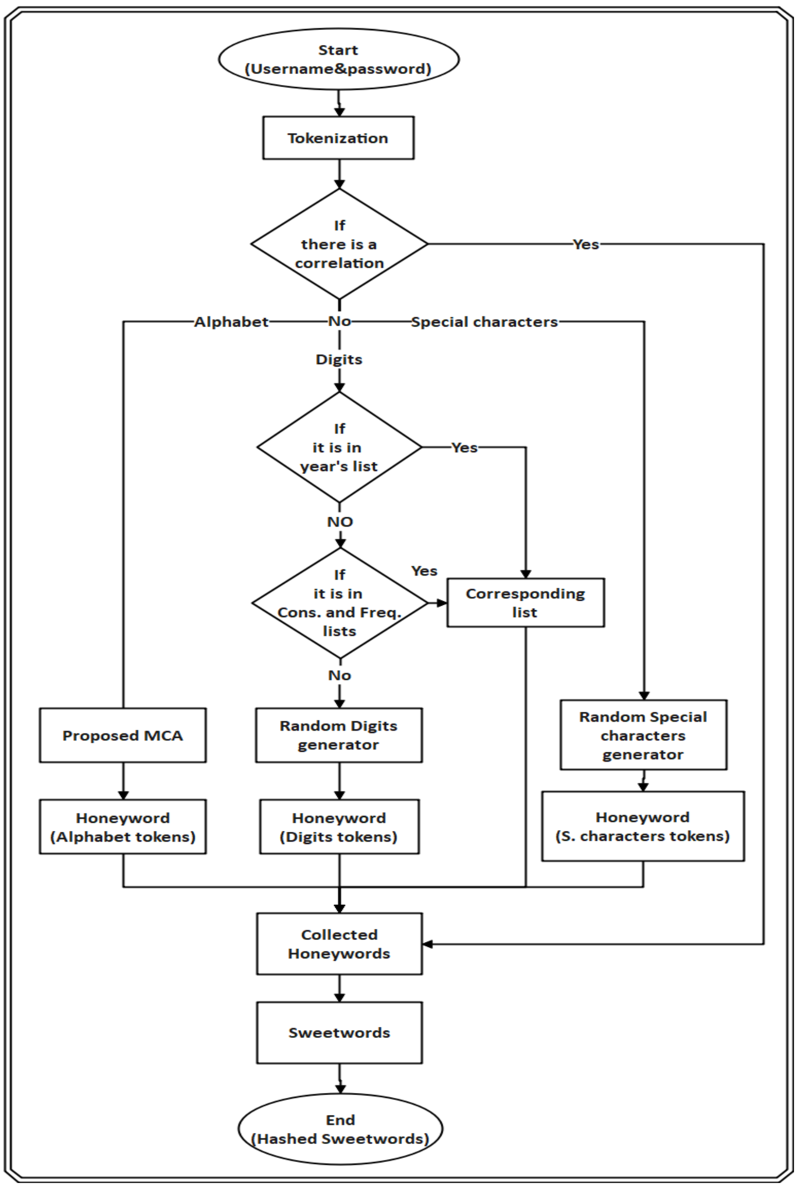

Fig. 1. Block-diagram for the proposed system

6. Proposed honeyword tokens generating algorithm

The proposed system has been used for three honeyword tokens generating algorithms. The proposed meerkat clan algorithm is used in this study to generate solutions as honeyword tokens. The proposed system is working on three parallel parts – alphabet, digits, and special characters generating. The first part will use the proposed MCA, the other two parts will use simple random token generating algorithms.

6.1. The proposed MCA alphabet tokens generator

This part considers as the most important part of the honeyword because it is a preferred target for the adversary to guess the real password. The proposed MCA is used as alphabet generating algorithm; the input for the proposed MCA is the alphabet token of the sugarword. It is considered as the seed that will be processed by the algorithm to generate the alphabet tokens for the honeywords. Algorithm 2 exhibits the generic steps of the proposed MCA alphabet tokens generator.

Algorithm 2. The generic steps of the proposed MCA alphabet token generator

a. Set the clan size (population size), foraging size, care size, worst foraging rate, worst care rate, max-generation, and evaluation criteria.
b. Generate the initial population (clan) randomly.
c. Evaluate the fitness of the population.
d. Make the best token as the sentry.
e. Divide the rest of the population into two groups; foraging and care.
f. Generate neighbors for the foraging group.
g. Choose the best neighbor as the foraging(*j*).
h. Swap the worst tokens of the foraging group with the best tokens of the care group.
i. Drop the worst of the care group tokens and generate ones randomly.
j. Evaluate the population.
k. chooses the best foraging as best-foraging.
l. If the best-foraging is better than the sentry then changes.
m. Update the best *a* tokens of the population in a buffer.
n. Repeat d. to m. until max-generation.

Take the best six tokens of the algorithm, make seven copies for each of the six alphabet tokens, and consider the 42 tokens as six groups (columns). Each group has seven similar tokens. Add seven copies of the alphabet seed. Thus, the algorithm will have 49 alphabet tokens. Example 1 shows a case for the honeyword alphabet token generating.

Algorithm 3. The pseudocode of the proposed MCA alphabet token generating
Parameters:

n	clan size (pop-size)
f	foraging size, where $n>f$
c	Care size $n-f-1$
Nt	Neighbor tokens
Wfr	Worst foraging rate
Wcr	Worst care rate
Mg	Max-generation
Ec	Evaluation criteria
a	best individual size

```
Begin
    Generate clan (n) of n tokens randomly
    Compute the fitness of the token’s clan
    Sentry=best token of the clan
    Divide the rest of the clan into two groups (foraging and care)
    For i=1 to Mg
        For j=1 to f
            Generate Nt neighbor tokens from the foraging group
            Foraging(j)= best token from Nt neighbors
        end for
        Swap the worst Wfr tokens of the foraging group with the best
     tokens of the care group
        Drop the worst Wcr tokens of the care group and generate ones
     randomly
        Compute the fitness of the token’s clan
        Best_foraging= best one of foraging group
        If best_foraging <= sentry then
            Sentry= best_foraging
        end if
        Update the best a individuals of the clan
    end for
End
```

6.2. The random digits tokens generator

The seed for the digits generating algorithm will be the digit token of the sugarword. The system will use the digits tokens generating algorithm to generate six tokens that have the same length as the seed. Take the six digits tokens and make seven copies for each one and consider the 42 tokens as six groups (rows) each group has seven similar tokens. Add seven copies of the digit seed, thus the algorithm will have 49 digits tokens.

Algorithm 4. The generic steps of the digits tokens generating algorithm

a. Set the number of the generated digits tokens *d*.

b. Changes the digits *dp* of the digit seed by randomly chosen digits.

6.3. The random special characters tokens generator

The seed for the special characters generating algorithm will be the special characters token of the sugarword. The system will use the special characters generating algorithm to generate six tokens that have the same length as the seed. Take the six special characters tokens and make seven copies for each one and consider the 42 tokens as six groups (rows) each group has seven similar tokens. Add seven copies of the digit seed. Thus, the algorithm will have 49 special characters tokens.

Algorithm 5. The generic steps of the special characters token generating algorithm

a. Set the number of the generated special characters tokens *s*.

b. Changes the special characters *sp* of the special characters seed by randomly chosen special characters.

Example 1. For the sugarword "6!cookie", the seed token of the alphabet is "cookie", of the digits is "6", and of the special characters is "!". The system will collect the generated tokens to be as follow:

6!cookie	6!cockle	6!cooking	6!cooker	6!coffee	6!crozier	6!cosine
3[cookie	3[cockle	3[cooking	3[cooker	3[coffee	3[crozier	3[cosine
8&cookie	8&cockle	8&cooking	8&cooker	8&coffee	8&crozier	8&cosine
4/cookie	4/cockle	4/cooking	4/cooker	4/coffee	4/crozier	4/cosine
8<cookie	8<cockle	8<cooking	8<cooker	8<coffee	8<crozier	8<cosine
1&cookie	1&cockle	1&cooking	1&cooker	1&coffee	1&crozier	1&cosine
0%cookie	0%cockle	0%cooking	0%cooker	0%coffee	0%crozier	0%cosine

The proposed system will randomly permutate the positions of sweetwrods and send the index of the user *u*(*i*) and the index of the sugarword *c*(*i*) to the honeychecker. Then, hashing and store sweetwords.

6.4. Proposed neighbors generating

The neighbors generating for the alphabet tokens depends on four operations, for each seed token four tokens will be generated then the best token of these tokens will take the place of the seed token, the four operations are:

a) Insert: Choose some characters' positions on the token randomly, and then choose random characters to insert.

b) Delete: Choose some characters' positions on the token randomly, and then delete it.

c) Translocation: Choose some characters' positions on the token randomly, and then exchange their positions.

d) Swap: Choose some characters' positions on the token randomly, and then choose random characters to swap.

6.5. Proposed evaluation criteria

The objective function for the proposed MCA will take the sum of the proposed evaluation criteria values as a metric to evaluate the generated solutions (alphabet tokens). The evaluation for the alphabet token sets a metric on the seed token that has been received from the sugarword. This research propose an evaluation criterion for the generated alphabet tokens, it is called the approximation factor. The value of the approximation factor is between 0 and 1, which is calculated as the sum of the four criteria values. Each criterion has a different value as mentioned in Section 7.1. Parameters, the four proposed criteria are:

a) The similarity in characters: The similarity in characters between the characters of the seed token and the generated token.

b) The similarity in length: The similarity in length between the characters of the seed token and the generated token.

c) The similarity in Part Of Speech (POS): The similarity in POS between the seed token and the generated token.

d) Meaningful word: The token is an English word or not?

7. Results and discussions

This portion of the research looks at: the parameter values, the experimental results, and a comparison between the proposed honeyword generating method and the previous methods; the parameter values, the experimental results, and a comparison between the proposed honeyword generating method and the previous methods.

7.1. Parameters

The proposed system uses many important parameters, which affect the performance of the system. The parameters used in the proposed system are shown in Table 1.

Table 1. The proposed system parameters values

No	Parameter	Value
1	Population Size *n*	90
2	Sentry	1
3	Foraging group size *f*	*n*/2
4	Care group size *c*	(*n*/2) – 1
5	Generated neighbor tokens size Nt	4
6	Ratio of changing in alphabet token during neighbor generating	0.3×(Token-length)
7	Max-generation Mg	30
8	Worst foraging rate size Wfr	0.2*n*
9	Worst care rate size Wcr	0.2*n*
10	evaluation criteria Ec	
	Similarity in characters	0.2
	Similarity in length	0.1
	Similarity in POS	0.1
	Meaningful word	0.6
11	Number of the generated alphabet tokens *a*	6
12	Number of the generated digits tokens *d*	6
13	Number of digits that changed in generated token *dp*	(Token-length)
15	Number of the generated special characters tokens *s*	6
14	Number of special characters that changed in generated token *sp*	(Token-length)

The proposed system uses many values for the parameters until choosing the values that provide the best performance for the proposed MCA. The parameters with executed values are:

- Population Size *n*: The study uses many population sizes (30, 50, 70, 90), the generation of size (90) was chosen.

- Sentry: Only one sentry is chosen.
- Foraging group size *f*: There are many sizes used (*n*/3, *n*/2, 2*n*/3), the foraging group size *n*/2 was chosen.
- Care group size *c*: There are many sizes used ((*n*/3) – 1, (*n*/2) – 1, (2*n*/3) – 1), the care group size *n*/2 was chosen.
- Generated neighbor tokens size Nt: Only four neighbors were generated.
- The ratio of changing in alphabet token during neighbor generating: The changing in token during neighbor generating is attempted in many sizes (one character, two characters, 0.25×(token length), 0.3×(token length), 0.5×(token length)), the changing size 0.3×(token length) was chosen.
- Max-generation Mg: Many iterations were used (10, 20, 30, 40,…, 100), there were no enhancements in results after 30 iterations. Thus, the max-generation number (30) was chosen for the alphabet token.
- Worst foraging rate size Wfr: Many sizes were used 0.1*n*, 0.2*n*, 0.3*n*, the Wfr size 0.2*n* was chosen.
- Worst care rate size Wcr: Many sizes were used 0.1*n*, 0.2*n*, 0.3*n*, the Wcr size 0.2*n* was chosen.
- Evaluation criteria Ec: For the evaluation criteria (the similarity in characters, the similarity in length, the similarity in Part Of Speech (POS), and meaningful word) consecutively, many values were used (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) & (0.2, 0.2, 0.1, 0.5) & (0.2, 0.1, 0.1, 0.6). The values (0.2, 0.1, 0.1, 0.6) were chosen because they result in meaningful words being generated.
- The number of the generated alphabet tokens a: only six alphabet tokens were generated.
- The number of the generated digits tokens d: only six digits tokens were generated.
- The number of digits that changed in generated token dp: The change in seed token during honeyword tokens generating was equal to the seed length.
- Set the number of the generated special characters tokens *s*: Only six special characters tokens were generated.
- The number of special characters that changed in generated token ds: The change in seed token during honeyword tokens generating was equal to the seed length.

Note: Each value chosen for a parameter is determined according to the results effectiveness monitoring of 10 executions to each value.

7.2. Experimental results

The experimental results are divided into two sections, the first focus on the individual token, and the second focus on the complete password. The proposed system was tested on many password tokens, especially the alphabet token, which is considered the most important token because the main aim for the attacker is to guess the real password.

7.2.1. Results for individual generated tokens (Table 2)

Table 2. Experimental results of the proposed system (alphabet, digit, special characters tokens)

No	Seed token	Pop-size/ Max-gen	Honeyword tokens/Approximation factor					
			Token1	Token2	Token3	Token4	Token5	Token6
1	**flower**	30/30	flown/ 0.916	woofer/ 0.9	flow/ 0.9	plow/ 0.866	floe/ 0.866	flywhee/ 0.85
		50/30	flown/ 0.916	smoker/ 0.9	flow/ 0.9	booker/ 0.9	floe/ 0.866	flew/ 0.866
		70/30	blower/ 0.966	roomer/ 0.9	flow/ 0.9	floor/ 0.883	floe/ 0.866	flaw/ 0.866
		90/30	plower/ 0.966	glower/ 0.966	blower/ 0.966	flown/ 0.916	fodder/ 0.9	flow/ 0.9
2	monkey	30/30	motley/ 0.933	hockey/ 0.933	convey/ 0.933	mona/ 0.8666	mink/ 0.866	honey/ 0.85
		50/30	donkey/ 0.966	mickey/ 0.933	conker/ 0.933	monk/ 0.9	fonteyn/ 0.9	manky/ 0.883
		70/30	donkey/ 0.966	mocker/ 0.933	mickey/ 0.933	monte/ 0.916	monk/ 0.9	mickey/ 0.9
		90/30	donkey/ 0.966	mickey/ 0.933	conker/ 0.933	jockey/ 0.933	pinkeye/ 0.9	monk/ 0.9
3	love	30/30	lode/ 0.95	hove/ 0.95	cove/ 0.95	hole/ 0.9	cole/ 0.9	lox/ 0.875
		50/30	wove/ 0.95	lome/ 0.95	dove/ 0.95	cove/ 0.95	pose/ 0.9	lox/ 0.875
		70/30	wove/ 0.95	lore/ 0.95	lope/ 0.95	lome/ 0.95	loge/ 0.95	lode/ 0.95
		90/30	lore/ 0.95	lope/ 0.95	lone/ 0.95	lome/ 0.95	loge/ 0.95	lode/ 0.95
4	sunshine	90/30	sunshade/ 0.95	outshine/ 0.95	shinbone/ 0.875	dauphin/ 0.862	puniness/ 0.85	dushanbe/ 0.85
5	**princess**	90/30	primness/ 0.95	prince/ 0.924	principal/ 0.9	princeton/ 0.9	mindless/ 0.875	huntress/ 0.875
6	dragon	90/30	aragon/ 0.966	craton/ 0.933	drag/ 0.9	diagonal/ 0.9	cracow/ 0.9	drab/ 0.866
7	lion	90/30	zion/ 0.95	pion/ 0.95	lyon/ 0.95	loon/ 0.95	limn/ 0.95	lien/ 0.95
8	sea	90/30	zea/ 0.933	yea/ 0.933	tea/ 0.933	spa/ 0.933	sew/ 0.933	sep/ 0.933
9	soccer	90/30	saucer/ 0.933	rocker/ 0.933	locoer/ 0.933	docker/ 0.933	bocce/ 0.916	docage/ 0.9
10	**Cheese**	90/30	Creese/ 0.966	Chouse/ 0.933	Chaise/ 0.933	Chess/ 0.916	Cheer/ 0.916	Cheerio/ 0.9
11	qpzmg	90/30	cpus/ 0.82	apex/ 0.82	cps/ 0.799	cobol/ 0.799	cool/ 0.779	pig/ 0.759
12	inemcr	90/30	iceman/ 0.9	inexact/ 0.871	incus/ 0.85	iver/ 0.833	ibidem/ 0.833	inchworm/ 0.825
13	rknus	90/30	sinus/ 0.919	runs/ 0.919	ramus/ 0.919	pinus/ 0.919	genus/ 0.919	janus/ 0.919
14	48	N/A	17	69	37	92	52	09
15	583	N/A	562	841	727	069	287	743
16	4369	N/A	2442	0996	4533	8101	7157	5047
17	!*	N/A	|+	{?	'%	%<	@]	}+
18	*@/	N/A	,{$	$<$	@}@	}+;	['''	_~{
19	*).!	N/A	'|-'	.[?@	&'~"	~-=\	?(}}	\!#]

This section presents an experimental result for some honeyword tokens that are generated by the proposed system, 13 alphabet, three digits, and three special characters tokens, which are presented in Table 2. The first 10 alphabet tokens are meaningful words and the next three alphabet tokens are rubbish words. As mentioned in Section 7.1. Parameters, the proposed MCA choose the Pop-size=90/Max-gen=30 as parameters values for the alphabet token, however, the first three tokens were tested in four different Pop-size (30, 50, 70, 90) with fixing of Max-gen=30. Results show that the four Pop-size lead to good results but the Pop-size=90 provide the better approximation factor. Note that the final return of the algorithm are the six best tokens, each token is accompanied by its approximation factor. Regarding the digit and special characters tokens, a simple random generator is used, the changes in characters is random but with the same seed token length, the final return of the algorithm are six tokens.

7.2.2. Results for complete generated honeyword (Table 3)

Table 3. Experimental results of proposed MCA (alphabet token) of Example 2

No	Attempts	Honeyword tokens/Approximation factor					
1	First	**famine/ 0.933**	**facile/ 0.933**	**damply/ 0.933**	**famulus/ 0.9**	**fandom/ 0.866**	**fame/ 0.866**
		fairy/ 0.85	smoker/ 0.9	fairway/ 0.842	zama/ 0.833	fail/ 0.833	milldam/ 0.785
		fatally/ 0.771	fails/ 0.75	camails/ 0.742	same/ 0.733	daily/ 0.716	
2	Second	**damply/ 0.933**	**farina/ 0.9**	**frail/ 0.883**	**fatal/ 0.883**	**fame/ 0.866**	**fafnir/ 0.866**
		armory/ 0.866	feminism/ 0.85	pali/ 0.833	famish/ 0.833	fail/ 0.833	fafnir/ 0.833
		familiar/ 0.8	familial/ 0.8	talk/ 0.799	miry/ 0.766	deadly/ 0.766	
3	Third	**famine/ 0.933**	**gamble/ 0.9**	**famulus/ 0.9**	**flail/ 0.883**	**sami/ 0.866**	**fagin/ 0.85**
		failure/ 0.842	mali/ 0.833	fnma/ 0.833	famish/ 0.833	fain/ 0.833	fail/ 0.833
		balmy/ 0.816	affix/ 0.816	mail/ 0.799	calamity/ 0.799	bail/ 0.799	milldam/ 0.785
		facially/ 0.75	balmily/ 0.714	–	–	–	–

In this section, many of the results for complete generated honeyword (alphabet, digits, and special characters tokens) are presented; these results are introduced as examples and appended with desirable properties of the proposed system-generated honeywords. The desirable properties of the generated honeywords are:

- Independent tokens are generated: The proposed system generates each token type of the password independently then collects them all. Example 1 in Section 6 shows the generating process for each token and the collection of them.
- Many solutions are generated: The generated honeywords of the proposed system do not provide only six tokens; a different number of honeywords are generated even if the Pop-size/Max-gen are fixed on 90/30. Example 2 shows how the proposed system can generate a different number of honeywords for the same sugarword and Pop-size/Max-gen.

- Different solutions are generated: The proposed system generates different honeywords in each generating process even if the Pop-size/Max-gen is fixed on 90/30. Example 2 shows how the proposed system can generate different honeywords for the same sugarword and Pop-size/Max-gen.

Example 2. For the sugarword family37(&) the example will show all generated honeyword alphabet tokens that exceed 0.6 approximation factor in four attempts, the Pop-size/Max-gen are fixed on 90/30. As shown in Table 3, there are different honeywords with different numbers for each of the attempts.

- **Different password patterns handling.** The system being proposed can handle the different password patterns of tokens order. In Example 1, the pattern was (digit, special characters, alphabet tokens).
- **Sweetwords high security.** The honeywords generated by the proposed system have high security against adversary guessing. The probability of picking the sugarword at random is 2%. Plus, the low probability of adversary right guessing, the sugarword in the proposed system cannot be guessed even if the adversary knows one of the sugarword tokens. In sweetwords of the proposed system, there are always seven sweetwords that have the same tokens. If the adversary knows one of the sugarword tokens, then he has the probability of picking the sugarword at random 1/7(≈14%). Example 1 shows that each token in sweetwords is redundant seven times.
- **Capital letters handling.** The proposed MCA can handle the capital letters of alphabet tokens, if there are capital letters in the sugarword then capital letters will be present in honeywords. The word “Cheese” in Table 2 shows alphabet tokens with capital letters.

7.3. Comparisons

The proposed system demonstrates the proposed MCA to be superior to the previous honeyword generation methods. This section compares the proposed MCA and the previous generating methods in three comparison dimensions, improving the honeyword generating process, enhancing the honeyword properties, and solving the issues of previous methods (Table 4).

- **First comparison dimension.** Improving the honeyword generating process.

The honeyword generating process of the proposed MCA has a main difference from the previous generating methods. The proposed MCA performs a real generating for password tokens depending on real password tokens as a seed then processes it to generate honeyword’s tokens derived from that seed. On the other hand, the previous generating methods do not not perform a generating process. They just change part of the password, suggest honeywords, ask the user to provide honeywords, use another user’s password, provide honeywords from published list passwords, or ask the user to provide personal information that creates honeywords.

- **Second comparison dimension.** Enhancing the honeyword properties.

Table 4. Comparison in most important honeyword properties

No	Methods	Flatness	DoS resistance	Storage
1	Proposed MCA	Unconditionally	Strong	No overhead
2	Chaffing-by-tail-tweaking [17]	Conditionally	Weak	No overhead
3	Chaffing-by –tweaking-digits [17]	Conditionally	Weak	No overhead
4	Simple model [17]	Conditionally	Strong	No overhead
5	Modeling syntax [17]	Conditionally	Strong	No overhead
6	Chaffing with “tough nuts” [17]	N/A	Strong	Overhead
7	Take-a-tail [17]	Unconditionally	Strong	No overhead
8	Random pick [17]	Conditionally	Strong	No overhead
9	Hybrid generation methods [17]	Conditionally	Strong	No overhead
10	Storage-index [18]	Conditionally	Weak	Overhead
11	PDP [19]	Conditionally	Strong	Overhead
12	Evolving password model [20]	Conditionally	Strong	No overhead
13	User-profile model [20]	Conditionally	Weak	Overhead
14	Append-secret model [20]	Conditionally	Strong	No overhead
15	User information method [21]	Conditionally	Weak	Overhead
16	Dictionary attack method [21]	Conditionally	Weak	No overhead
17	Generic password list method [21]	Conditionally	Strong	No overhead
18	Shuffling characters method [21]	Conditionally	Weak	No overhead

The proposed MCA attempts to enhance the properties of the honeywords that are not always present in an optimum way in the previous honeyword generating methods. The most important honeyword properties are as follow.

- **Flatness.** This is the probability that the adversary can succeed in guessing the correct password among several false passwords. Under certain conditions, all approaches can reach (1/k) perfect flatness. The condition is that nothing makes the correct password distinct from the fake words, both in the real password and its relationship to the username. The proposed MCA provides perfect flatness unconditionally; on the other hand, most previous generating methods provide perfect flatness under some conditions. Satisfying of some conditions to achieve perfect flatness is a disadvantage, while not needing to satisfy any conditions is an advantage. For the first honeyword system of J u e l s and R i v e s t [17] the probability that the adversary can succeed in guessing the sugarword was 1/20=5%, the proposed system has a less probability with 1/49(≈2%). In all previous honeyword systems, the adversary can guess the sugarword if he knows one of its tokens. On the other hand, the adversary cannot guess the sugarword even if he knows one of its tokens; this is so because each token of the proposed system sweetwords is redundant seven times. If the adversary knows one of the sugarword tokens, he has a 1/7(≈14%) chance to pick the sugarword.
- **DoS resistance.** The system can make honeywords that cannot be guessed by the adversary. The DoS attack performed by entering a honeyword and then bring to denial the services of the system. The proposed MCA provides honeywords that

cannot be guessed by the adversary even if he knows the sugarword in some way. On the other hand, many of the previous generating methods provide honeywords that may be guessed by the adversary.

- **Storage.** The proposed MCA stores only username and sweetwords, on the other hand, many of the previous generating methods store extra information and details.

Table 4, shows a comparison between the proposed MCA and previous generating methods in the most important honeyword properties:

- **Third comparison dimension.** Solving the issues of previous methods.

The previous honeyword generation methods faced many issues. The proposed MCA solves the most important seven issues of honeyword systems. The seven issues are:

- **Conditional flatness issue.** This is the satisfying of some conditions to achieve perfect flatness, which is considered a weakness, while unconditional flatness means that there is no need to satisfy any condition, considered a strength. The proposed MCA provides perfect flatness unconditionally, on other hand, the most of previous generating method provides perfect flatness under some conditions.
- **Weak DoS resistance issue.** This is the existence of possibility for the adversary to guess the honeywords, while strong DoS resistance means that the adversary can not guess the honeywords. The proposed MCA has a strong Dos resistance; on the other hand, many of the previous generating methods have a weak Dos resistance.
- **Storage overhead issue.** This is the demand for extra storage cost. The proposed MCA does not need extra storage cost, while many previous generating methods need extra storage cost.
- **Correlation issue.** One of the concerns is the existence of a correlation between username and password. So, the correct password may simply be identified from honeywords. Therefore, the proposed MCA lets the correlated part stay as is in the honeywords.
- **Consecutive or frequent numbers issue.** Because users prefer rememberable number patterns, many choose to use consecutive or frequented digits in their passwords, such as ‘123’, ‘1234’, ‘111’ or ‘2222’ then this leads to recognizing the sugarword. So, the proposed system provides a list of the most used consecutive and frequented number patterns. If there are consecutive or frequent numbers appearing in the sugarword, the system will randomly choose numbers from the list to the honeywords.
- **Special date issue.** Many users tend to choose a number concerning the birth date, anniversary, the best year in their study, or any other similar date to include in their passwords that will disclose the sugarword. Therefore, the proposed system will provide a list of the last 50 years. Then the system will randomly choose years from the list to the honeywords if the year’s digits appeared in sugarword.
- **User information security issue.** Many of the previous generating methods use the technique that leans on personal knowledge-based questions, forcing users to provide personal information and detail to help the methods to generate honeywords. If the system is compromised and personal information disclosed, this information

may be used on another system and pose a threat to the user. Thus, using this technique constitute a security issue considering it as a weakness, while not using it is a strength. Hence, the proposed MCA does not ask the user for any personal information.

Table 5 shows a comparison between the proposed MCA and previous generating methods in the most important honeyword system issues.

Table 5. Comparison in most important honeyword system issues

No	Methods	Conditional flatness issue	Weak DoS resistance issue	Storage overhead issue	Correlation issue	Consecutive and frequent numbers issue	Special date issue	User information security issue
1	Proposed MCA	No	No	No	No	No	No	No
2	Chaffing-by-tail-tweaking [17]	Yes	Yes	No	Yes	Yes	Yes	No
3	Chaffing-by – tweaking-digits [17]	Yes	Yes	No	Yes	Yes	Yes	No
4	Simple model [17]	Yes	No	No	Yes	No	No	No
5	Modeling syntax [17]	Yes	No	No	Yes	Yes	Yes	No
6	Chaffing with “tough nuts” [17]	N/A	No	Yes	No	N/A	N/A	No
7	Take-a-tail [17]	No	No	No	No	No	No	No
8	Random pick [17]	Yes	No	No	Yes	No	No	No
9	Hybrid generation methods [17]	Yes	No	No	Yes	Yes	Yes	No
10	Storage-index [18]	Yes	Yes	Yes	Yes	No	No	No
11	PDP [19]	Yes	No	Yes	No	Yes	No	No
12	Evolving password model [20]	Yes	No	No	Yes	Yes	Yes	No
13	User-profile model [20]	Yes	Yes	Yes	Yes	Yes	No	Yes
14	Append-secret model [20]	Yes	No	No	No	Yes	No	No
15	User information method [21]	Yes	Yes	Yes	Yes	Yes	No	Yes
16	Dictionary attack method [21]	Yes	Yes	No	Yes	No	No	No
17	Generic password list method [21]	Yes	No	No	Yes	No	No	No
18	Shuffling characters method [21]	Yes	Yes	No	Yes	Yes	Yes	No

8. Conclusion

This study provides a novel approach for the honeyword generating process. The new method lies on the meerkat clan intelligence algorithm, a metaheuristic swarm intelligence algorithm. It has undergone several changes to fit the problem space and has produced the solutions as honeywords. As a result, the proposed system successfully harnesses an intelligent algorithm (meerkat clan intelligence algorithm) for security purposes, precisely password cracking detection (honeyword system).

The proposed system succeeds in benefiting from MCA’s properties in solutions generating (proximity principle, quality principle, diverse response principle, stability principle, and adaptability principle) to generate honeywords. The proposed MCA improves the honeyword generating process, enhances the honeyword properties, and solves the issues of previous methods. The most important and complex token of the sugarword is the alphabet token, so the proposed system is used, including the proposed MCA to generate the alphabet token. On the other hand, the digit and special characters tokens can be generated by a simple random generating algorithm.

The experimental results show the algorithm’s success in generating passwords in all its tokens (alphabet, digits, and special characters), especially the alphabet token, despite the difficulty of this token as it can be related to meaningful words. The alphabet token generating provided great results for generating meaningful words from meaningful words. Most interesting is that the algorithm is able to find meaningful words from rubbish words. As a result of analysis, the proposed system concludes that the Pop-size should be bigger than the Max-gen, by experience, the proposed system chooses Pop-size=90/Max-gen=30. The results show many desirable properties for the generated honeywords (independent tokens generating, many solutions generating, different solutions generating, different password patterns handling, sweetwords high security, and capital letters handling)

The comparisons between the proposed MCA and the previous generating methods shows the superiority of the modern method in three dimensions: improving the honeyword generating process (through depending on the real password for honeywords generating), enhancing the honeyword properties (through providing perfectly flat honeywords, strong DoS resistance, and a moderate amount of storage), and solving the issues of previous methods (through providing the solutions for the most important seven issues of honeyword systems). The most important properties which is the flatness showing a great superiority of the proposed system. For the first honeyword system of J u e l s and R i v e s t [17] the perfect flatness is (1/20=5%), the proposed system has a better flatness with 1/49(≈2%). Furthermore, the adversary has a 1/7(≈14%) chance to pick the sugarword even if he knows one of the sugarword tokens.

This article suggests recommendations for honeyword generation methods by taking advantage of the experience provided by this study of using metaheuristic algorithms and attempts to find another intelligent technique that may provide optimal solutions (honeywords). Further research in this field could be directed towards finding other issues facing honeywords systems and attempting to solve them.

R e f e r e n c e s

1. M u k t h i n e n i, V., R. M u k t h i n e n i, O. S h a r m a, S. J. N a r a y a n a n. Face Authenticated Hand Gesture Based Human Computer Interaction for Desktops. – Cybernernetics and Information Technologies., Vol. **20**, 2020, No 4, pp. 74-89.
2. M. Lehto, P. Neittaanmäki, Eds. Cyber Security: Analytics, Technology and Automation. – Cham, Springer International Publishing, Vol. **78**. 2015.

3. G e n ç, Z. A., S. K a r d a ş, M. S. K i r a z. Examination of a New Defense Mechanism: Honeywords. – In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). G. P. Hancke, E. Damiani, Eds. Cham, Springer International Publishing. Vol. **10741**. 2018, pp. 130-139.
4. K u s u m a, A. B., Y. R. P r a m a d i. Implementation of Honeywords as a Codeigniter Library for a Solution to Password-Cracking Detection. – IOP Conf. Ser. Mater. Sci. Eng., Vol. **508**, May 2019, No 1, p. 012134. DOI: 10.1088/1757-899X/508/1/012134.
5. W i n, T., K. S. M. M o e. Protecting Private Data Using Improved Honey Encryption and Honeywords Generation Algorithm. – Adv. Sci. Technol. Eng. Syst., Vol. **3**, 2018, No 5, pp. 311-320. DOI: 10.25046/aj030537.
6. C h a k r a b o r t y, N., S. M o n d a l. Towards Improving Storage Cost and Security Features of Honeyword Based Approaches. – Procedia Comput. Sci., Vol. **93**, 2016, No September, pp. 799-807. DOI: 10.1016/j.procs.2016.07.298.
7. W a n g, R., H. C h e n, J. S u n. Phoney: Protecting Password Hashes with Threshold Cryptology and Honeywords. – Int. J. Embed. Syst., Vol. **8**, 2016, No 2-3, pp. 146-154. DOI: 10.1504/IJES.2016.076108.
8. P a l a n i a p p a n, S., V. P a r t h i p a n, S. S t e w a r t K i r u b a k a r a n, R. J o h n s o n. Secure User Authentication Using Honeywords. – Lecture Notes on Data Engineering and Communications Technologies, Vol. **31**, 2020, pp. 896-903.
9. H o m a y o u n i, S. M., D. B. M. M. F o n t e s. Metaheuristic Algorithms – Metaheuristics for Maritime Operations. Hoboken, NJ, USA, John Wiley & Sons, Inc., 2018, pp. 21-38.
10. T e z e l, B. T., A. M e r t. A Cooperative System for Metaheuristic Algorithms. – Expert Syst. Appl., Vol. **165**, 2021, No May 2020, p. 113976. DOI: 10.1016/j.eswa.2020.113976.
11. T o s h e v, A. Particle Swarm Optimization and Tabu Search Hybrid Algorithm for Flexible Job Shop Scheduling Problem – Analysis of Test Results. – Cybernernetics and Information Technologies, Vol. **19**, 2019, No 4, pp. 26-44.
12. K u m a r, A., D. K u m a r, S. K. J a r i a l. A Review on Artificial Bee Colony Algorithms and Their Applications to Data Clustering. – Cybernernetics and Information Technologies, Vol. **17**, 2017, No 3, pp. 3-28.
13. G r e e n, D., A. A l e t i, J. G a r c i a. The Nature of Nature: Why Nature-Inspired Algorithms Work. – Model. Optim. Sci. Technol., Vol. **10**, 2017, pp. 1-27. DOI: 10.1007/978-3-319-50920-4_1.
14. A l-O b a i d i, A. T. S., H. S. A b d u l l a h, Z. O. A h m e d. Meerkat Clan Algorithm: A New Swarm Intelligence Algorithm. – Indonesian Journal of Electrical Engineering and Computer Science, Vol. **10**, 2018, No 1. pp. 354-360. DOI: 10.11591/ijeecs.v10.i1.
15. A b d A l r a d h a A l s a i d i, S. A., D. K. M u h s e n, S. M. A l i. Improved Scatter Search Algorithm Based on Meerkat Clan Algorithm to Solve NP-Hard Problems. – Period. Eng. Nat. Sci., Vol. **8**, 2020, No 3. DOI: 10.21533/pen.v8i3.1563.
16. J a m e e l, N., H. S. A b d u l l a h. A Proposed Intelligent Features Selection Method Using Meerkat Clan Algorithm. – J. Phys. Conf. Ser., Vol. **1804**, February 2021, No 1, p. 012061. DOI: 10.1088/1742-6596/1804/1/012061.
17. J u e l s, A., R. L. R i v e s t. Honeywords: Making Password-Cracking Detectable. – In: Proc. of 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS'13), 2013, No October 2015, pp. 145-160. DOI: 10.1145/2508859.2516671.
18. E r g u l e r, I. Achieving Flatness: Selecting the Honeywords from Existing User Passwords. – IEEE Trans. Dependable Secur. Comput., Vol. **13**, March 2015, No 2, pp. 284-295. DOI: 10.1109/TDSC.2015.2406707.
19. C h a k r a b o r t y, N., S. M o n d a l. On Designing a Modified-UI Based Honeyword Generation Approach for Overcoming the Existing Limitations. – Comput. Secur., Vol. **66**, 2017, pp. 155-168. DOI: 10.1016/j.cose.2017.01.011.
20. A k s h i m a, A., D. C h a n g, A. G o e l, S. M i s h r a, S. K. S a n a d h y a. Generation of Secure and Reliable Honeywords, Preventing False Detection. – IEEE Trans. Dependable Secur. Comput., Vol. **5971**, 2018, No c, pp. 1-13. DOI: 10.1109/TDSC.2018.2824323.
21. A k i f, O. Z., A. F. S a b e e h, G. J. R o d g e r s, H. S. A l-R a w e s h i d y. Achieving Flatness: Honeywords Generation Method for Passwords Based on User Behaviours. – Int. J. Adv. Comput. Sci. Appl., Vol. **10**, 2019, No 3, pp. 28-37. DOI: 10.14569/IJACSA.2019.0100305.

22. B r i n d t h a, J., K. R. H i t h a e i s h i n i, R. K o m a l a, G. A b i r a m i, U. A r u l. Identification and Detecting of Attacker in a Purchase Portal Using Honeywords. – In: Proc. of 3rd IEEE Int. Conf. Sci. Technol. Eng. Manag. (ICONSTEM'17), Vol. **2018-Janua**, 2017, pp. 389-393. DOI: 10.1109/ICONSTEM.2017.8261414.
23. G e n ç, Z. A., G. L e n z i n i, P. Y. A. R y a n, I. V a z q u e z S a n d o v a l. A Critical Security Analysis of the Password-Based Authentication Honeywords System under Code-Corruption Attack. – Communications in Computer and Information Science, Vol. **977**, 2019, pp. 125-151.
24. G e n ç, Z. A., G. L e n z i n i, P. Y. A. R y a n, I. V. S a n d o v a l. A Security Analysis, and a Fix, of a Code-Corrupted Honeywords System. – In: Proc. of 4th International Conference on Information Systems Security and Privacy, Vol. **2018-Janua**, 2018, No Icissp, pp. 83-95. DOI: 10.5220/0006609100830095.
25. C a t u o g n o, L., A. C a s t i g l i o n e, F. P a l m i e r i. A Honeypot System with Honeyword-Driven Fake Interactive Sessions. – In: Proc. of 2015 Int. Conf. High Perform. Comput. Simulation (HPCS'15), 2015, pp. 187-194. DOI: 10.1109/HPCSim.2015.7237039.
26. N a t h e z h t h a, T., V. V a i d e h i. Honeyword with Salt-Chlorine Generator to Enhance Security of Cloud User Credentials. – Commun. Comput. Inf. Sci., Vol. **746**, 2017, pp. 159-169. DOI: 10.1007/978-981-10-6898-0_13.
27. M o e, K. S. M., T. W i n. Improved Hashing and Honey-Based Stronger Password Prevention against Brute Force Attack. – In: 2017 International Symposium on Electronics and Smart Devices (ISESD'17), Vol. **2018-Janua**, October 2017, pp. 1-5. DOI: 10.1109/ISESD.2017.8253295.

Received: 02.12.2021; Second Version: 20.12.2021; Accepted:11.01.2022 (*fast track*)