ISSN: 2582-8118 ***Volume 2, Issue 2; March 2022***

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYSIS IN SCIENCE AND ENGINEERING

Web: https://www.iarj.in/index.php/ijrase/index

2. A More Robust and Secure Version of Paired Distance Protocol

Neetu, Mohona Ghosh
Indira Gandhi Delhi Technical University for Women, Delhi, India.

Sweta Mishra
Shiv Nadar University, Greater Noida, Uttar Pradesh.

ABSTRACT

In 2013, Ronald L. Rivest and Ari Juels proposed the concept of honeywords for the early detection of password breaches. Thereafter, many other honeyword generation approaches have been proposed in the literature. However, all of these existing honeyword approaches require a very large amount of space to store k-1 no. of honeywords along with one real password for each user. Hence, these existing approaches are not memory efficient.

In 2015, Paired Distance Protocol (PDP) was proposed by Nilesh et al. which was the first and only algorithm to overcome this limitation of large memory footprint and was shown to be as secure as its peers. In this work, we show that the design of the PDP protocol has many limitations which makes it insecure against several new attacks.

Thus, despite being memory efficient, it cannot be rendered fit for practical use owing to reduced security. We then propose an improved design framework of Paired Distance Protocol (PDP) which overcomes all the existing limitations and is resistant to all the attacks.

We provide a detailed security analysis of our proposed version and also perform a comparative analysis with the original PDP protocol and other peer algorithms existing in the literature. We show that our proposed version retains memory efficiency as its predecessor and at the same is more secure than the original PDP, thus making it better than the other approaches which is verified through provided analysis.

KEYWORDS:

Authentication, Password, Honeyword, Paired Distance Protocol, Detection technique, Password breach detection.

1. Introduction:

Password data breach is a big problem in today's world where online users are increasing day by day either professionally or personally. Many people still have poor password choosing habits, which makes hackers easier to hack data. A recent study says that the password "123456" is used by 23 million accounts [1]. The biggest challenge in the security domain is to detect database breaches automatically when such incidents happen. Although attacks cannot be stopped completely, only the impact of an attack can be reduced. Therefore, several authors provide different solutions to this early breach detection problems.

In 2013, Ronald et al. [11] proposed a new concept known as "Honeywords" which was very much appreciated because the existing systems can be updated easily to this new approach. In this, honeywords also known as fake passwords are added along with the real passwords in the system database. Another auxiliary server is known as "Honeychecker" stores the index of the real passwords. Honeychecker is capable of raising alarm when any suspicious login attempt is detected depending on the policy set by the administrator. Several honeyword approaches have been proposed by Ronald et al., for more details you can refer [11]. All these generic honeyword approaches store k-1 honeywords along with one real password for each particular user which requires a large amount of space. Therefore, these techniques are not memory efficient which is one of the biggest limitations.

Chakraborty et al. [6] came with the solution to overcome the storage overhead problem and proposed a storage efficient Paired Distance Protocol (PDP). Despite being memory efficient, PDP was not secure. It fails to provide security against multiple system intersection attack as discussed by Akshima et al. in [8], which put down the memory efficiency factor of PDP. Therefore, we propose a secure version of Paired Distance Protocol which is resistant to multiple system intersection attack and provides high security against DoS attack, brute force attack, dictionary attack, password guessing attack with negligible overhead on storage space. In this modified version of PDP, we have added a system generated secret string which makes our propose PDP highly secured as discussed in security analysis in section 4. We show a comparative analysis of the propose version of PDP with the original PDP and generic honeyword approaches which proves that the propose version of PDP is more secure than the other without hiding the orientation of characters in hcl unlike original PDP.

1.1 Motivation and Contribution:

Generic honeyword approaches proposed by Ronald et al. [11] were not memory efficient. Space is an important factor, as online users are increasing day by day. This honeyword concept also gives rise to multiple system intersection attack discussed by Akshima et al. in paper [8]. Suppose if a user chooses the same password on different systems and the attacker gets access to multiple systems, and then he can easily distinguish the real password from the honeywords via intersection. Generic honeyword approaches fails to provide security against multiple system intersection attack except for Tweaking by tail [11]. Tweaking by tail approach adds a random string at the end of the original password. This forces users to remember different passwords for different systems, which is practically not possible. However, tweaking by tail also does not resolve the storage issues.

Chakraborty et al. proposed a storage efficient Paired Distance Protocol. Despite being memory efficient, PDP was not secure. PDP also has certain limitations discussed in section 3. We require a solution that is highly secured as well as memory efficient. Thus, from the existing literature survey, we gist the following motivations behind this work, which can be summarized as:

1. Despite being memory efficient, PDP fails to provide security against Multiple System Intersection attack as discussed in paper [8]. This drives us to think of a solution that provides security as well as efficiency.
2. In the original PDP [7], the author uses password strength meter concept to ensure randomness between the password and random string. In reality, we cannot rely on a password meter. This motivates us to provide a solution that can introduce high randomness even if the user chooses a simple password and random string.
3. In paper [7], authors suggest encryption techniques to encrypt hcl and share encrypted hcl among multiple systems over insecure channels since the orientation of characters is kept secret which increases overhead. For example, if PDP approach is used by many organizations then generating a large no of keys, is a cumbersome task to do. Therefore, there is a need for a more robust protocol which achieves high security without making use of the encryption algorithm.

After being motivated by the above given facts, we have made the following contribution in this paper.

Contribution 1: We propose an improved version of PDP, which is highly secured as well as memory efficient. This enhanced version of PDP addresses all the above-mentioned issues that exist in the original PDP.

Contribution 2: We show that hcl need not be encrypted in our propose version of PDP, unlike the original PDP.

Contribution 3: We provide a detailed security analysis of the improved version of PDP. We also show the comparative analysis of the improved version of PDP with the original PDP and the existing approaches.

Roadmap: This paper is categorized into different sections. Section 2 covers the overview and limitations of the original PDP approach in detail. Section 3 describes the improved version of PDP. In Section 4 we provide the security analysis of the proposed PDP approach. In Section 5 we compare our proposed version of PDP the original PDP and other generic honeyword approaches. Finally, we conclude our work in Section 6.

2. Overview of Original Paired Distance Protocol and its Limitations:

In 2015, Chakraborty et al. [6] proposed a storage efficient Paired Distance Protocol (PDP). PDP uses a honey circular list of 26 elements in total which comprised of alphabets and digits that is securely stored in the main server. Along with the username and password, the system stores the distance chain derived from random string RS chosen by the user in Password File Fp in the main server separated by "-". Here, the distance chain is the paired distance between any two characters of the random string.

In earlier approaches, honeychecker maintains an index of the correct password. In PDP approach, Honeychecker stores the username along with the first character of RS. At the time of registration, user enters a random string along with username and password. During authentication, system derives the paired distance of the entered random string if password entered by user is correct. If the derived paired distance gets matched with the paired distance stored in server, system communicates the first character of random string. If first character of the submitted random string gets matched with the stored one in honeychecker, then honeychecker directs positive feedback to the admin. Generic honeyword approaches store k-1 honeywords on the system internally which requires a large amount of space. Moreover, honeywords generated by generic honeyword approaches are so close to the password due to which legitimate users can do typing mistake which is also a big problem. However, in PDP, there is no such concept of generating honeywords. Therefore, no chance of typing mistakes by a legitimate user. For more detail you can refer paper [6].

Username u_i	**Password** $H(pwd_i \| RS_i)$	**Paired Distance of Random String** RS_i
u_1	$H(pwd_1 \| RS_1)$	RS_1
u_2	$H(pwd_2 \| RS_2)$	RS_2
u_3	$H(pwd_3 \| RS_3)$	RS_3
.	.	.
.	.	.
.	.	.
u_n	$H(pwd_n \| RS_n)$	RS_n

Table 1. Example of Password File Fp

Username u_1	**First Character** fc_i
u_1	fc_1
u_2	fc_2
u_3	fc_3
.	.
.	.
.	.
u_n	fc_n

Table 2. File stored in Honeychecker

Despite being memory efficient, PDP was also not secure. There also exist some security flaws in PDP which have been discussed below:

Multiple System Intersection Attack: It is a human tendency to set the same password for multiple systems. In paper [8], Akshima et. al has shown that multiple system attack is possible in PDP.

Multiple System Intersection considering Output Attack: Suppose among m different systems, the same circular list is shared and the same RS is chosen for different accounts. Assuming both these entries are available to the attacker, the intersection will reveal the actual password on which he needs to work on to calculate the distance chain and interpret the paired distance stored with the password. An attacker needs to perform maximum n trials to calculate all combinations of RS in which one is correct. Here n represents the total no of elements in hcl. An adversary needs to target only one system and determine all combinations for that particular system and then try all these combinations on another system. Doing this he will get hit on one system at least when the first character is not known. Hence, it does not prevent MSIO attack.

Multiple System Intersection considering Input Attack: Suppose a different circular list is shared among m different systems and the same random string RS is chosen. Attacker just needs to find all sugarwords combinations on one system after gaining access to both the entries. Then trying these sugarwords on other systems, attacker will find a match for the password on either system and successfully implements the MSII attack.

DoS Attack: In PDP, DoS attack is possible if repetition of characters is allowed in RS as discussed in paper [6]. Suppose the user chooses RRR and attacker may submit SSS as the distance chain is the same as the RRR to perform DoS attack. DoS attack is possible even after keeping hcl secret. Password can be easily compromised via brute force attack or dictionary attack or social engineering attack as the password hash is stored as a single entity in password file Fp along with the random string in the main server.

Randomness property: Another limitation is that success of the original PDP depends on the randomness of the string. If there exists a co-relation between password and RS then randomness is low (e.g. password-rab, RS- bit, or RS is dictionary password rose). In paper [7], author has suggested to use password meter concept to check randomness. For example, password "Helloworld @1" gives three different result on 3 different sites [3],[2],[4]. One gives very strong, second gives strong and third gives weak as a result when checked on password meter. Therefore, it is recommended to choose RS which should be strong enough and completely random.

Overhead of using Encryption techniques to share hcl: In paper [7], the author uses the encryption technique to share hcl among multiple systems over insecure channels to maintain the privacy of hcl by not revealing the orientation of characters to anyone. This increases the overhead of generating a large no of keys if the same approach is used by different organizations.

Therefore, after analyzing all these facts we propose a new version of PDP which overcomes all the limitations discussed above.

3. Proposed Paired Distance Protocol:

This approach uses different concept which enriches our proposal in different aspects from the original PDP. 3

1. Extra elements in hcl: First, we have incorporated some special characters in our hcl which are generally used in making a password. Our hcl contains 44 elements in total which consists of 26 alphabets (a to z), 10 digits (0 to 9) and 8 special characters ('@','#','\','%','!','*','&','_'). However, you may include more i.e., ('|',':','-','^','~'), we have included mostly used once. In original PDP, there were 26 elements in total.

The main motive behind increasing the sample space of hcl is to ensure randomness as the larger the no of elements in hcl, the more would be the choices. This also increases the complexity or effort required to compromise a system. Honey circular list will be stored in the main server along with the password file. In proposed PDP, there is no need to hide the orientation of characters in hcl unlike original PDP which is an added advantage.

2. System generated secret string: Second, we propose to add a system generated secret string corresponding to each user. This secret string will be stored in honeychecker along with username, first character of the random string as shown in Table 4. The length of the secret string l_{SS} may vary maximum from 3 to 12 character. However, it is recommended to use 3 to keep storage space as low as possible. In our analysis, we show that we have achieved high security by adding a secret string of length 3. In original PDP, password is stored as single unit in password file fp. In the newly propose version of PDP, we are storing the hash of the concatenated input, i.e. password, random string RS, secret string SS i.e., H(pwd||RS||SS) in place of hashed password in the password file Fp as shown in Table 3. The addition of secret string makes this approach strongly resistant to DoS attack, brute force attack, dictionary attack, password guessing attack as discussed in security analysis in section 4.

Username u_i	**Password** $H(pwd_i\|\|RS_i\|\|SS_i)$	**Paired Distance of Random String** RS_i
u_1	$H(pwd_1\|\|RS_1\|\|SS_1)$	RS_1
u_2	$H(pwd_2\|\|RS_2\|\|SS_2)$	RS_2
u_3	$H(pwd_3\|\|RS_3\|\|SS_3)$	RS_3
.	.	.
.	.	.
.	.	.
u_n	$H(pwd_n\|\|RS_n\|\|SS_n)$	RS_n

Table 3. Example of Password File Fp

Username u_1	**First Character** fc_i	**Secret String** SS_i
u_1	fc_1	SS_1
u_2	fc_2	SS_2
u_3	fc_3	SS_3
.	.	.
.	.	.
.	.	.
u_n	fc_n	SS_n

Table 4. File stored in Honeychecker

3.1 Working Principle of Proposed PDP approach:

Registration Process: During registration, the user will be asked to enter a username, password and Random String RS of length 2 to 4 characters. As soon as the user register after providing all this 3 above mentioned information, the system generates a secret string SS of length l_{SS}. To keep storage requirements as low as possible, we would recommend to use l_{SS} as 3.

The system calculates the hash of the concatenated input i.e, password, random string, and secret string H (pwd||RS||SS). We are not storing password hash only as a single entity because of the high chances of being compromised. In the proposed PDP, the username, H (pwd||RS||SS) and paired distance will be stored on the main server. Honeychecker will store the username, the first character of random string RS, and the system generated secret string SS.

Authentication Process: During Authentication, the user will provide a user name, password, and random string RS. Then system derives the paired distance between the random strings submitted by the user. If the calculated paired distance matches with the paired distance stored in the password file, it communicates the first character to the honeychecker.

It then verifies the first character of the submitted with the first character in the honeychecker. If it matches, then the system picks the system generated secret string SS stored in honeychecker and calculates the hash of the submitted password, random string, and secret string. If the calculated hash gets matched with the hash which is stored in the Password File Fp, then the honeychecker directs the Positive feedback to the system otherwise directs a negative response to the system.

3.2 Rationale behind adding this secret string:

The reason behind adding the system generated secret string has been discussed below:

In the original PDP, the password was directly getting stored as a single entity in the password File Fp in a hashed form as shown in Table 1. The addition of system generated secret string SS makes this approach resistant to multiple system intersection attack even if the user chooses same password or random string among the different system. In original PDP, the intersection attack can give you the targeted password in hashed form, which can be easily cracked by brute force or dictionary or social engineering attack. This in turn, makes it easy to guess random string easily if both these are co-related. It is human tendency to use such string which can be easily remembered. In Proposed PDP, we are storing H (pwd||RS||SS) which makes each password different even if chosen same password or random string, therefore intersection attack would not leak the targeted information.

The addition of secret string SS and random string RS along with the password and then storing the hash of all 3 ensures high randomness. If there exists a co-relation between the chosen RS and password, the addition of a secret string makes password completely random. In the original PDP, honeychecker was storing the username and first character of the RS shown in Table 2. In the proposed PDP, we are storing the system generated secret string along with the username and first character as shown in Table 4, assuming honeychecker is a highly secured system.

3.3 Desirable length of secret string:

In our analysis, we have shown that we are able to achieve high security when the length of the secret string is 3 by calculating the probability of attack. For example, DoS attack probability is discussed in the security analysis section when length 1,2 or 3. For default value of parameters |hcl| =44, l_{SS} =3 which we have recommended, the probability of mounting DoS attack in the proposed version is 0.00054 which is very less. For l_{SS} =1, the probability of mounting DoS attack in proposed version is 0.977. For l_{SS} =2, the probability of mounting DoS attack comes out to be 0.227. The probability is very less when the length of the secret string is 3 rather than 1 or 2. However, it can vary from 3 to 12 characters, but to keep storage as low as possible, we recommend it to 3.

4. Security Analysis of the Proposed PDP approach:

In this section, we analyze our proposed PDP approach on few security parameters which are used for evaluating the robustness of any honeyword generation techniques- 1) Security against MSV 2) DoS resistant 3) Brute force 4) Dictionary attack 5) Password guessing attack. Along with these, we have also taken some other parameters into account i.e., Typo-safety, storage cost, randomness.

1. Security against Multiple System Vulnerability: In this proposed PDP approach, Multiple System Input/output attack [8] is not possible. As discussed above in section 4, in MSIO and MSII attack, the user needs to target only one system after getting password matches in different systems. An adversary can crack the password by applying brute force

or dictionary attack and can also calculate all the combinations of RS after cracking first. In this proposed approach we are not storing the password as a single entity. We are adding a secret string here which makes this approach highly resistant to MSIO and MSII attack. In the main server we are storing the hash of 3 elements, i.e., password, random string RS, and secret string SS as H (pwd||RS||SS). An adversary cannot calculate the password without knowing the secret string. Assuming the honeychecker is a highly secured system, an adversary will never get the secret string generated by the system as it is completely random.

2. DoS Resistant: DoS attack is a type of attack in which an adversary may intentionally create a misconception that password file Fp has been com- promised just to trigger a strong response. An overly sensitive system may force to block all accounts globally. Our Proposed PDP approach stores the hash of a password, random string, and secret string H (pwd||RS||SS). An attacker needs to enter a password for crossing the first level of authentication. Stored hash is highly resistant to inversion attack. It is not possible to calculate the same input value that can produce the same hash. An attacker can only calculate all combinations of RS but he doesn't have any idea about the password and the secret string which is not enough to crack the password. In paper [7], the author has shown that the probability of guessing random string for mounting DoS attack was 0.028 without knowing the orientation of characters. Since we have not kept the orientation of characters in hcl secret, an attacker can calculate all combinations of RS. Therefore, the probability Pr_{SS} of mounting DoS attack can be calculated from the below equation:

$$Pr_{SS} = \frac{(|hcl|-1)}{P_{l_{SS}}^{|hcl|}} \qquad (1)$$

Where |hcl| is the the total no of elements in the honey circular list =44 and l_{SS} is the length of the secret string. Here the denominator represents the probability of guessing a secret string. For default value of parameters |hcl| =44, l_{SS} =3, the probability of mounting DoS attack in proposed version is 0.00054 which is very less than the original PDP. For l_{SS} =1, the probability of mounting DoS attack in proposed version is 0.977. For l_{SS} =2, the probability of mounting DoS attack comes out to be 0.227. The probability is very less when the length of secret string is 3 rather than 1 or 2. That's why we recommended it to 3. Assuming, if anyhow attacker knows the password by shoulder surfing attack [13] [10][14]. However, the chances are very less for this as everyone is aware now of cyber frauds. It is a targeted attack in which a particular person is targeted only. This is not possible in all scenarios; only a few can be compromised but not all. The countermeasure is that the user must be aware, he should use a strong password following strong password policies [9]. In those scenarios, he may use different passwords for different systems.

3. Security against Brute Force attack: Brute Force attack is the process of submitting a large no of co-related passwords in search of guessing original Password [5], [12]. Honeypot account plays important role in improving security against brute force attack. The honeypot accounts are nothing just fake accounts that are created by the system itself in early detection of brute force attack. If an attacker submits a password from honeypot accounts, then the system can easily detect that the password file Fp has been compromised. In the proposed approach honeypot accounts will be created in a way whose passwords can be easily guessed by the attacker. After guessing the password, he can get all the combinations of the RS as hcl is public. Now he can try all the combinations one by one.

He needs to do n trials out of which 1 will be correct. Here n is the total no of elements in hcl. For these honeypot accounts, we will not fix the number of login attempts. As soon as he will try each RS combination with the cracked password and for the correct password and RS, the secret string will be appended. An adversary will be able to successfully logged into the system. Now the system will take action accordingly as per the policies set by the admin. For example, the admin my sets off an alarm or notifies the system admin or let login proceed as usual on honeypot system tracing the source of login carefully.

Let the total number of accounts in a system be N including hpt honeypot accounts. The probability of hitting at least one honeypot accounts in k trials can be calculated from the given equation.

$$Pr_{hit} - Honeypot = 1 - \left(\frac{N - h_{pt}}{N}\right)^{k} \qquad (2)$$

For some value of the total account, N= 1000000, honeypot account Hpt= 20,000 (2 percent of total account), the probability of hitting at least one honeypot account after trials k=800 would be 0.99 which is good.

4. Security against Dictionary attack: Traditional approach uses the concept of "salting" when a user chooses the same password for multiple sites. Hence, a random salt value is added to the password. The drawback is that Salt is stored in the clear text along with the hash which can be easily compromised by dictionary attack. Our proposed PDP approach is resistant to dictionary attack also. In the proposed PDP approach, the introduction of a new component as a secret string makes our proposed approach highly resistant to dictionary attack.

The secret string cannot be compromised which is stored in honeychecker assuming honeychecker is a secured system. Passwords cannot be easily compromised as we are not storing the password alone in the system. Here password is stored as H (pwd||RS||SS) which is non-deterministic as the secret string cannot be compromised. Assuming password of length 8, a secret string of length 3 to keep storage as low as possible, and a random string of length 3 and all these 3 entities composed of elements that have been included in hcl i.e., 44 elements in total, then the efforts required to successfully implement DoS attack is:

Effort required = $44^8 * 44^3 * 44^3$

Which is computationally infeasible. This makes this our proposed approach resistant to dictionary attack also.

5. Password guessing attack: Most of the users employ their personal information to build password tail. Even if an adversary will guess the correct RS, but he cannot guess the correct password as we have incorporated an extra element SS that is system generated secret string. Adversary cannot invert the hash i.e., H (pwd||RS||SS) stored in password File Fp without knowing the secret string SS for calculating password. Password guessing attack is not possible in our proposed approach.

6. Typo-Safety: A honeyword generation algorithm is said to be typo-safe if the typing mistake does not lead to a negative response by honeychecker. In original PDP, honeychecker lead to a negative response in 2 cases a) if a user enters subpart of RS wrong, in that case, the system will never evaluate the distance chain or, 2) if a user enters all elements of RS wrong by typing mistake, the probability that same distance chain will be generated can be calculated from below equation:

$$Pr_{Typo-Safe} = |hcl| - 1 * \sum_{i=0}^{l_{RS}-1} \frac{1}{|hcl|-1} \quad (3)$$

Where hcl is the total no of elements the in the honey circular list and l_{RS} is the length of the random string. In original PDP, for default value of parameters l_{RS} =3, |hcl|=36, the probability was $0.81 * 10^{-3}$ as per author in paper [6]. In proposed PDP, for default value of parameters l_{RS} =3, |hcl| =44, the probability is $0.54*10^{-5}$ which is less than the original PDP. Therefore, the chances of typing errors are very less than the original PDP.

7. Storage cost: The storage cost required by the existing honeyword approaches discussed in paper [11] were very large. Then PDP approach comes into the picture. Despite being memory efficient, the original approach was not secure. Our proposed approach is a secured variant of the original PDP approach. In this approach, we have added an extra component named as secret string SS generated by the system. To keep storage space as low as possible, we would recommend 3. This puts negligible overhead on the storage space. We have also provided a storage cost comparison between the proposed approaches over the existing approaches in Table 6.

8. Randomness property: The success of the original PDP depends on the randomness of the string as discussed in section 3. Therefore, it is recommended to choose RS that is strong enough and completely random which is not possible in real scenarios. However, a random string is chosen by the user only, we cannot control it fully, but in the proposed PDP approach, even if there is a co-relation between the password and random string, password would be completely random as we are storing the hash of 3 elements i.e., H(pwd||RS||SS). Adversary needs the password to guess the random string which he cannot predict without the knowledge of the new component i.e., secret string which ensures high randomness.

5. Comparison of the Proposed PDP approach with original PDP and other generic honeyword approaches.

In Table 5, we provide a comparison and show that our proposed version of PDP is highly secured than the original PDP. The Proposed PDP is resistant to multiple system intersection attack and provides high security against DoS attack, brute force attack, dictionary attack, password guessing attack with negligible overhead on storage space which is an improvement over the original PDP.

No adversary can get access into the system without the knowledge of secret string which is almost impossible to even predict. In Table 6, we compare the proposed PDP approach with the generic honeyword approaches.

Parametres	Original PDP	Proposed PDP
MSIO resistant	✓	✗
MSII resistant	✓	✗
DoS resistant	weak(probability=0.028)	strong (probability=0.00054
Typo-Safety	weak(probability=$0.81*10^{-3}$)	strong (probability=$0.54 * 10^{-5}$)
Randomness	Low	high
Brute force attack resistant	weak(probability=0.98)	strong(probability=0.99)
Dictionary attack resistant	weak(probability=26^8*26^3)	high(probability=$44^8*44^3*44^3$)
Password guessing resistant	Weak	strong
storage cost	$2MB_U + MB_H + MB_k$ where $MB_k = l_{RS} + l_{fc}$	$2MB_U + MB_H + MB_k$ *Where* $MB_k = l_{RS} + l_{fc} + l_{SS}$

Table 5: Comparison between original PDP and proposed PDP. Here MB_U, MB_H refers to the length of username, password, l_{RS}, refers to the length of random string, l_{fc} refers to length of first character and l_{SS} refers to length of secret string all in bytes.

6. Conclusion:

For a honeyword approach to be more effective, it should be highly secured as well as memory efficient. The original PDP approach was memory efficient but not secured. Our proposed approach is an improvement over the original PDP approach. In this work, We show that by introducing a secret string called SS in the proposed approach, we have addressed all the limitations of the original PDP approach. Hence, we have achieved high security as well as efficiency in terms of memory.

Honeyword approaches	MSII resistant	MSIO resistant	DoS resistant	Flatness	Typo-Safety	Storage cost
Chaffing by Tweaking [11]	✗	✗	weak	< 1/k	✓	$2MB_U+kMB_H+MB_k$
Chaffing with password [11]	✗	✗	strong	n/a	✗	$2MB_U+kMB_H+MB_k$
Chaffing by tough-nuts [11]	✗	✗	strong	n/a	✗	$2MB_U+kMB_H+MB_k$
Take-a-tail [11]	✓	✓	weak	1/k	✗	$2MB_U+kMB_H+MB_k$

Honeyword approaches	**MSII resistant**	**MSIO resistant**	**DoS resistant**	**Flatness**	**Typo-Safety**	**Storage cost**
Close-num formation [11]	✓	✓	weak	1/k	✗	$2MB_U+kMB_H+MB_k$
Caps-key based [11]	✓	✓	weak	1/k	✗	$2MB_U+kMB_H+MB_k$
Modified-tail [11]	✓	✓	weak	1/k	✗	$2MB_U+kMB_H+MB_k$
Evolving password [8]	✗	✗	strong	1/k	✓	$2MB_U+kMB_H+MB_k$
user-profile model[8]	✗	✗	moderate	1/k	✓	$2MB_U+kMB_H+MB_k$
append-secret model[8]	✓	✓	strong	1/k	✓	$2MB_U+kMB_H+MB_k$
Proposed PDP	✓	✓	strong	n/a	✓	$2MB_U+ MB_H + MB_k$

Table 6: Comparison between existing honeyword approaches and propose version of PDP. Here MB_U, MB_H and MB_k refers to the length of the username, password and length of the index in bytes for existing approaches except propose PDP whose detail is given in Table 5. Here, Flatness is defined as the probability of guessing correct password from the honeywords. For further details, you can refer paper [11], [8].

References:

1. https://gadgets.ndtv.com/internet/news/worst-passwords-2020-nordpass-annual-report-123456-2327502
2. https://www.my1login.com/resources/password-strength-test/
3. https://www.security.org/how-secure-is-my-password/
4. http://www.passwordmeter.com/
5. Boˇsnjak, L., Sreˇs, J., Brumen,.: Brute-force and dictionary attack on hashed real-world passwords. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). pp. 1161–1166. IEEE (2018)
6. Chakraborty, N., Mondal, S.: A new storage optimized honeyword generation approach for enhancing security and usability. arXiv preprint arXiv:1509.06094 (2015)
7. Chakraborty, N., Mondal, S.: On designing a modified-ui based honeyword gener- ation approach for overcoming the existing limitations. Computers & Security 66, 155–168 (2017)
8. Chang, D., Goel, A., Mishra, S., Sanadhya, S.K.: Generation of secure and reliable honeywords, preventing false detection. IEEE Transactions on Dependable and Secure Computing 16(5), 757–769 (2018)
9. Eldar, A., Sharoni, I., Mendelson, T., Blumenthal, U.: Techniques for password attack mitigation (Mar 6 2012), uS Patent 8,132,018
10. Guo, Y., Zhang, Z., Guo, Y.: Superword: A honeyword system for achieving higher security goals. Computers & Security p. 101689 (2019).

11. Juels, A., Rivest, R L.: Honeywords: Making password-cracking detectable. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 145–160 (2013)
12. Kirkup, M.G., Little, H.A., Adams, N.P.: System and method for protecting a password against brute force attacks (Sep 16 2014), uS Patent 8,838,975
13. Kwon, T., Shin, S., Na, S.: Covert attentional shoulder surfing: Human adver- saries are more powerful than expected. IEEE Transactions on Systems, Man, and Cybernetics: Systems 44(6), 716–727 (2013)
14. Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: An underestimated threat. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 1242–1254 (2016).