Secure Password Storing Using Prime Decomposition

Chaovalit Somboonpattanakit and Nawaporn Wisitpongphan

***Abstract*—Nowadays, passwords are widely used for authentication and they are the main targets of many hackers. Over the past few years, there were several unfortunate major password leak incidents. Billions of passwords were exposed to public due to unsecured password storing methods. To further exacerbate the situation, many users prefer to use the same password on multiple websites or services. Hence, a password leak from one source will likely cause damage to users' data in many other systems. In this study, we proposed a secure method for storing passwords by incorporating prime decomposition technique. The novel Prime Decomposition Password Storing (PDPS) technique does not store original or encrypted password, but allows the same password to be converted differently on different systems without adding extra information. We claimed that the proposed PDPS can protect the confidentiality of the password in the event of password database leaked.**

***Index Terms*—password, password storing, prime decomposition, secure, authentication.**

I. Introduction

PASSWORD is apparently the most popular authentication method for modern applications. During system registration, users have to create a password that is easy to remember. Most systems force users to create a complex password, which is composed of numbers, alphabets, and sometimes special characters, to prevent conventional password attacking methods. However, users' ability to remember deteriorates as the rules for creating passwords get more complicated. As a result, many users tend to reuse the same password on different systems. Therefore, besides forcing the users to create a strong password, service providers have to also ensure that they provide a secure password storing system that can protect users' information in the event of password database leak.

There many research studies that focused on enhancing password storing security. The most popular technique is the use of hash functions [1], [2] or Key Stretching [3], [4]. Example of these techniques are MD5, SHA3, SCRYPT, ARGON2, etc.

However, using a pure hash function without additional protection mechanism such as "salt" for storing passwords makes this approach vulnerable to time-memory tradeoff password attack [5] using a pre-computed table, which is readily available on the Internet [6]. Therefore, adding salt will only make hacking more time and resource consuming, but cracking the salted hashed passwords is still plausible.

Manuscript received May 29, 2020; revised January 14, 2021.

C. Somboonpattanakit is a PhD candidate of King Mongkut's University of Technology North Bangkok, Bangkok, Thailand. He is now with Department of Information Technology, Rajamangala University, Bangkok, Thailand. (email: s5907011956126@email.kmutnb.ac.th)

N. Wisitpongphan is a Assistant Professor of Data Communication and Networking Department, King Mongkut's University of Technology North Bangkok, Bangkok, Thailand. (e-mail: nawaporn.w@itd.kmutnb.ac.th)

Key Stretching [7] is an adjustable hash function used for enhancing the security of the password stored in the database. The Key Stretching relies on high computing power and large memory usage during the hashing process, thus, cracking or reversing is infeasible. Although Key Stretching is the best mechanism for storing passwords, the principle of this method is still the same as a normal hash method with some additional salt in the plain text format.

Storing hashed or salted hashed passwords may no longer be secure, hence, we propose another way to securely store passwords. We claim that the proposed method will be safe even in the event of password leak.

II. Background

According to NIST disclaimer statement [8], most users reuse the same password on multiple systems when forcing them to use a complex password. In addition, if a user has selected a good password, the system should not force the users to change the password often because users may simply create a new password that is similar to the old one, e.g., adding a certain numeric or alphabet pattern at the end of the old password. Therefore, when the passwords got leaked to the public, the attacker can use those passwords to attack other systems. Therefore, forcing users to come up with a complex password does not always imply that the users' password will be secured. To get around this problem, we proposed a novel technique for securing the stored passwords. The main idea is to allow users to select any password they want without imposing strict set of rules or requirements. Users can use any password that is easy for them to remember.

A. Authentication mechanisms

Authentication is a mechanism that is used to identify a person by using a certain set of data. Typically, users have to provide various personal information to identify themselves on different systems for security purpose. The set of data that is commonly used to identify a user can come from these factors [9]:

1) Something you know: password, pin
2) Something you have: smart card, hardware token
3) Something you are: fingerprint, iris scan
4) Somebody who knows you: human relations

Authentication mechanisms can include one, two, or multiple factors. Using more than one factor helps increase the security level during the validation process but also add burden to users because it requires extra steps or additional devices, e.g., using both the password and the finger scan [10] or requesting the users to input the OTP (One-time password) sent to them via SMS and validate within a specified time [11].

B. Password Storing Techniques

Usually systems or application developers have various ways to protect the stored passwords [12]:

1) Plain text password: This is the most unsecured method for storing passwords because the password is stored as is without any encryption. Hence, attackers who get access to the password database can readily use the leaked password to exploit other systems. In 2016, roughly 100 million plain text passwords from the large social media website (vk.com) was exposed to public [13]. The lesson learned from such an event is that users should come up with unique password for each website.

2) Encrypted password: The password is encrypted with a certain algorithm such as AES, DES or 3DES prior to storing [14], [15]. The key for encrypting the password is also stored in the same database because it will be used again in the validation process. This method can keep the password secured by making the password unreadable. This, by far, is the most popular method because of its low complexity and a slightly higher security level than that of the plain text method. However, in the event of database leak, there is a possibility that the attackers can also obtain the key from the same database.

3) Hashed password: This method works by applying a one-way hash function such as MD5 or SHA3 to a password and stores the hash results in the database instead of storing the original passwords. When the attackers gain access to the database, they will see only hash values and will have to put extra effort to obtain the original passwords. However, this method is still vulnerable to the rainbow table attack [16].

4) Salted hash password: This method will add a random string to the password before hashing. The goal of adding salt to the password [17] is to create a different hashed value of the same password, thereby preventing Time-Memory Trade-Offs attack [18]. However, because the salt must be stored in the plain text format in the database, attackers who know the salt and the hashed value can still use this information to crack the password.

5) Key Stretching: This method enhances the one-way hash function by using adjustable parameters such as number of hash time, length of result, and encryption algorithm [19]. The added features either increase computational costs or increase memory usage. Therefore, when attackers have access to the password database, they cannot easily crack the passwords because of the extra security which requires them more time to crack.

C. Prime Decomposition

The prime number decomposition is a method of finding multiplication factors which are two prime numbers. This mathematical problem is computationally intensive for a very large number. Therefore, this prime decomposition principle is commonly used to design a well-known and widely used cryptography algorithm such as the RSA asymmetric key encryption that is widely used in the SSL Certificate and digital signature [20].

The integer factorization is a difficult problem in computer science [21]. Nowadays, there is no algorithm that can solve this problem in polynomial time using the classical computer. In this work, we applied this technique for storing the password in the database.

D. Related Work

One of the most popular methods for protecting the passwords is to use hash functions such as MD5 or SHA3 [22]. However, these methods are prone to Time-Memory Trade-Offs attack because of their fast hashing speed. Therefore, the traditional one-way hash function is not the best way to keep passwords secured, although in general it is still widely used. As a result, key stretching techniques are becoming more popular.

Unlike traditional hash functions, key stretching runs at a very slow speed and consumes significant amount of computer resources [3], especially the processing power, in order to prevent an attacker from using a precomputed table. However, as the processing speed of the GPU or FPGA hardware increases, the key stretching technique, which relies solely on the use of processing power, is no longer safe.

According to the aforementioned problem, there are several studies that resorted to other techniques for securing the password. Colin Percival [4] proposed memory-hard (SCRYPT) algorithm that uses significant amount of memory instead of extra processing power to prevent GPU or FPGA attack. This technique is more appealing because the memory space is typically limited and the advancement in hard drive development is usually slower than that of the processor. However, SCRYPT still has flexibility and complexity problems, thus, not practical for deploying in a real system. Despite these problems, this concept has become a model for many ideas later.

To overcome the challenge faced by memory consumption-based approach, LYRA2 algorithm [23] uses cryptographic sponge function, which allows flexibility in configuring the size of the input and output independently. By exploiting the properties of a sponge, i.e., absorbing and squeezing, the user can adjust the parameters to suit any working environment.

Among the techniques described in this section, the password hash function together with a key stretching is the most secured technique for storing the passwords. However, the most popular method is the plain text salt approach which is still vulnerable to password leak.

Therefore, we propose a password storing algorithm that exploits the difficulty of the prime number factorization to enhance the security of the passwords.

III. Prime Decomposition Password Storing

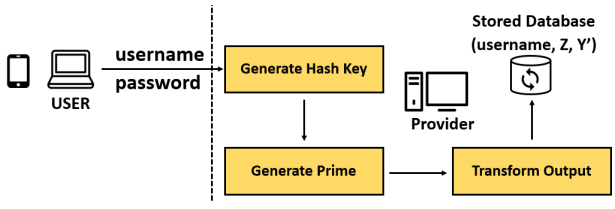

Fig. 1. PDPS Framework

In this section, we describe the proposed Prime Decomposition Password Storing (PDPS) algorithm in detail. The proposed technique uses mathematical problem which is hard to solve in a short amount of time.

A. Generating Hash Key from the Hashed Password

The challenge of creating algorithms for storing passwords is finding an appropriate password representation to be stored in the database for identifying users when accessing the system. This stored value must be confidential.

Fig. 2. Hash Key Generation Process

Therefore, our first step is to prevent the attackers from recovering the plain text passwords when gaining access to the password database. For this purpose, we use SHA3-512 to convert the users' passwords to a 512-bit hash key (HK) in hex format [24].

$$HK = \text{SHA3-512 (user's password)} \quad (1)$$

As an additional protection to the passwords, PDPS will not store the hash keys in the database directly. Instead, the result value, HK, will later be used as a parameter for adjusting the prime number in the later step.

B. Prime Number Generator

In this section, we find the optimal length of the prime number to be used in PDPS [41]. Similar to the strength of the private key and public key of RSA [25], [26], the longer the length of the prime number, the more secure the password.

However, the length of the prime number can adversely affect the performance of PDPS. Hence, to find out the optimal prime length, we measured the time taken to randomly create two large prime number of size 512, 768, 1024, 1536, and 2048 bits using OpenSSL version 1.1.1g [27] on Intel Core i7-7700HQ 2.80 GHz with 8GB RAM running 64-bit Window 10.

To generate a random prime number, we used Miller-Rabin test [42] described in Algorithm 1 because it is much faster than other primality tests and is easy to implement [43], [44]. In the experiment, we set Miller-Rabin to use the default number of iterations, i.e., k = 64, for testing prime number of different lengths. The experiment was repeated 1,000 times for each prime length.

Algorithm 1 Miller Rabin Algorithm

1. Find integers k, q, $k > 0$, $qodd$, so that $(n-1) = 2^k q$
2. Select a random integer a, $1 < a < n-1$
3. if $a^q \ mod \ n$ = 1 then return (maybe prime);
4. for j = 0 to $k-1$ do
5. if $a^{2^j q} \ mod \ n$ = $n-1$ then return (maybe prime);
6. return (composite);

According to the results shown in Figure 3, the time taken to create a pair of prime numbers ramped up quickly as the prime length exceeds 1,024 bits. That is the prime number creation time doubled as prime length increases from 1,024 to 1,536 bits: from 0.1637 seconds to 0.3268 seconds. Therefore, the optimal prime length chosen for this study is 1,024 bits.

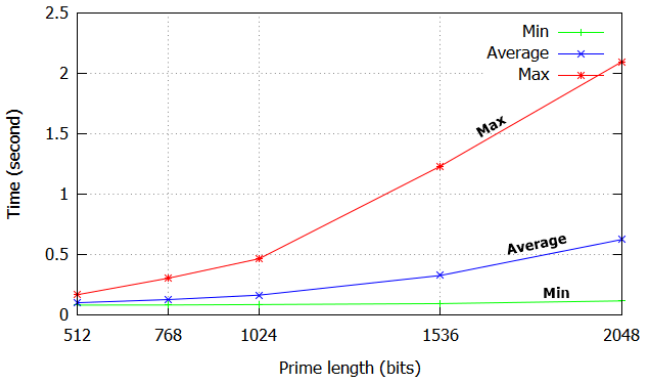

Fig. 3. Prime Number Generating Time

C. Prime Decomposition Password Storing

The goal of PDPS is to transform the same password into different values without using salt. The result of the transformation will then be stored in the database instead of storing the hashed password. Therefore, the transform function has to be hard enough for attackers to solve, i.e., either computationally intensive or time-consuming. Large Prime factorization is a hard problem which cannot be solved in a short amount of time. While this technique is quite common in cryptography, we explore how such technique can be used to protect the stored password.

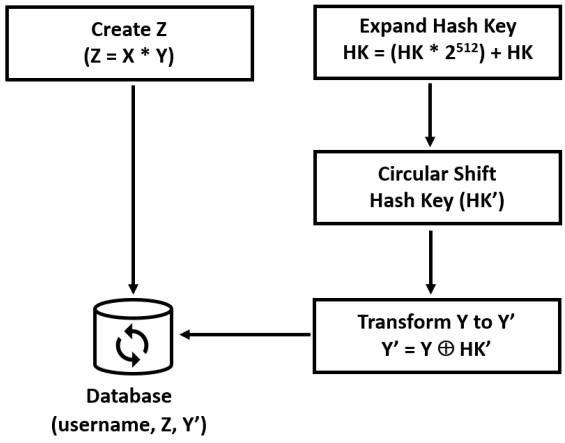

Fig. 4. Prime Decomposition Password Storing

According to Figure 4, PDPS generates an integer, Z, which is a multiplication of the two randomly generated 1024-bit prime numbers, X and Y, as is shown in Equation (2).

$$Z = random(X) \times random(Y) \quad (2)$$

The prime number, X, used in the calculation will only be used in this step to find Z and will not be stored in the database. The value of Y, on the other hand, will be transformed to Y' using substitution technique [29] by following these procedures:

1) Expanding the hash key (HK): Generate a 1024-bit HK by concatenating the hash key (HK) obtained from Equation (1) to itself. This can be done by using the following equation.

$$HK = (HK \times 2^{512}) + HK \tag{3}$$

2) Obtaining the circular shifted hash key (HK'): The substitution process can be done by right circular shifting each hexadecimal digit of the 1024-bit HK from step 1 by the length of the password to obtain the 1024-bit HK'. For example, Fig. 5 shows an example of a 10-digit shifted hash key when password length is equal to 10 characters.

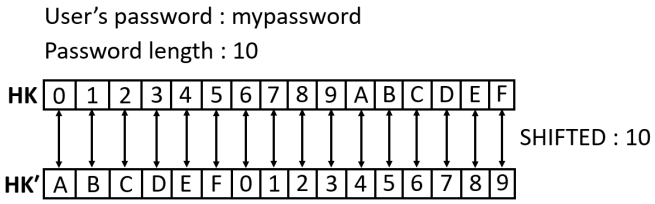

Fig. 5. Example of a hash key substitution process with the password length = 10

3) Transformation of the prime number Y: To secure a stored prime number Y, PDPS uses a bitwise exclusive OR operation on Y and the 1024-bit HK', as shown in Equation (4). The obtained Y' is stored in the database along with Z from Equation (2).

$$Y' = Y \oplus HK' \tag{4}$$

According to the Equation (2) – (4), our proposed PDPS uses a one-way transformation algorithm which is designed to prevent attackers from recovering the original hash key or password from Y' while at the same time allow for reverse calculation when the input is the correct password. Algorithm 2 describes the password transformation in detail while Table I summarizes the output of each password transformation steps in PDPS.

TABLE I
SUMMARY OF PASSWORD TRANSFORMATION

	Description	Output
1	Extracting a password length	Password's length
2	Converting a user's password into unreadable strings using SHA3	Hash Key (HK)
3	Expanding the hash key to 1024 bit by concatenate HK to itself	1024-bit HK
4	Converting 1024-bit HK to a hexadecimal format	256 hex character (HK)
5	Right circular shifting each hexadecimal byte of 256 character (HK) by the password length	256 hex character (HK')
6	Randomly generating the first 1024-bit prime number	X
7	Randomly generating a second 1024-bit prime number	Y
8	Transforming Y by XOR with HK' and store this value in the database	$Y \oplus HK' = Y'$
9	Multiplying X with Y and store this value to the database	$X \times Y = Z$

* Only Y' and Z are stored in the database for verification.

Algorithm 2 Password Transformation

Require: user's password
1. Extract user's password length
2. HK = SHA3(user's password)
3. HK_{1024} = $HK \cdot HK$
4. $HK_{1024decimal} \rightarrow HK_{256hex}$
5. Right Shifting HK' = $HK_{256hex} \leftarrow$ password length

Require: Generate X 1024 bit prime number
Require: Generate Y 1024 bit prime number
6. Y' = $Y \oplus HK'$
7. Z = $X * Y$
8. Store Y' and Z to the Database

D. Prime Decomposition Password Verification

Prime Factorization of an Integer is the main challenge of PDPS password verification process. The Z value is a product output between X and Y which were not stored in the database. Therefore, to check whether the entered password is a valid password, the Z value stored in the database has to be divisible by Y with no remainder. In addition, Y cannot be 1 or the same as Z value.

The first five steps of the verification process are the same as that of the transformation process. After completing these steps, the result obtained is an HK'. If a user enters a correct password the system will use this password to reverse Y' to Y by XOR with the HK' from the previous step. In the final step, the value Z in the database will be divided by Y. If a user enters a correct password, the result is the output with no remainder as shown in the Figure 6

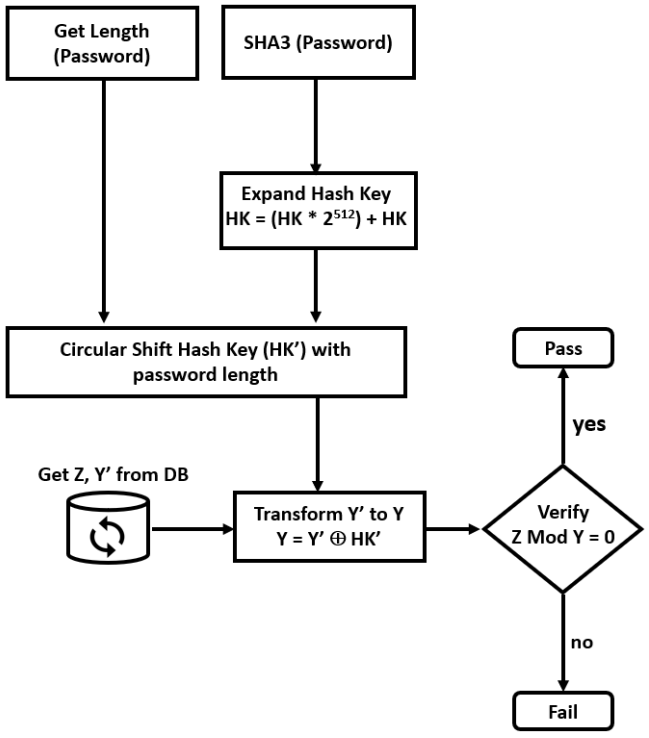

Fig. 6. Prime Decomposition Password Verification

The verification process can be described by Algorithm 3 and the output of each verification step is shown in Table II.

TABLE II
SUMMARY OF PASSWORD VERIFICATION PROCESS

	Description	Output
1	Extracting a password length	Password's length
2	Converting a user's password into unreadable strings using SHA3	Hash Key (HK)
3	Expanding the hash key to 1024 bit by concatenate HK to itself	1024-bit HK
4	Converting 1024-bit HK to a hexadecimal format	256 hex character (HK)
5	Right circular shifting each hexadecimal byte of 256 character (HK) by the password length	256 hex character (HK')
6	Reading Y' from the database	Y'
7	Calculate Y by $XOR\, Y'$ with HK'	$Y = Y' \oplus HK'$
8	Reading Z from the database	Z
9	Divide Z by Y	result = Z/Y

* Y in step 9 must not equal 1 or Z value.

Algorithm 3 Password Verification

Require: user's password
1. Extract user's password length
2. HK = SHA3(user's password)
3. HK_{1024} = $HK \cdot HK$
4. $HK_{1024decimal} \rightarrow HK_{256hex}$
5. Right Shifting HK'= $HK_{256hex} \leftarrow$ password length

Require: Y' from user's database
Require: Z from user's database
6. Y' = $Y \oplus HK'$

Ensure: $Y \neq 1$ and $Y \neq Z$
7. $result$ = Z / Y
if $result$ = 0 **then**
  return $true$
else
  return $false$
end if

IV. PERFORMANCE OF PDPS

In this section, we analyzes the performance of the proposed algorithm by considering speed, storage space, security, and tolerance.

A. Speed Analysis

The algorithm speed is defined as the time elapsed between the moment the password is entered to the system and the time that the access grant confirmation is received.The passwords used in this experiment were taken from a collection of 10,000 most popular passwords from xato.net [30]. In the experiment, we compared our proposed Prime Decomposition algorithm with 7 different techniques: Encryption, Normal Hash, Double Iteration Hash, Salted Hash, Double Iteration Hash, Key stretching and Double Iteration Key stretching. All eight sets of passwords, undergo different protection algorithms, will be stored in separate tables on the same machine with MySQL database management version 5.7.15.

Table III shows the important fact about each algorithm [31], [32], [33]

TABLE III
PASSWORD STORING ALGORITHM

Type	Notation	Algorithm	Salt	Parameter
Encrypted	E	aes-256	✗	✗
Normal Hash	NH	sha3(data)	✗	✗
Double Iteration Hash	DIH	sha3(sha3(data))	✓	✗
Salted Hash	SH	sha3(data+salt)	✓	✗
Double Iteration Salted Hash	DISH	sha3(sha3(data+salt))	✓	✗
Key stretching	KS	argon2(data+salt)	✓	✓
Double Iteration Key stretching	DIKS	argon2(argon2 (data+salt))	✓	✓
Prime Decomposition	PDPS	PDPS	✗	✓

Verification process of each of the 10,000 passwords was performed on a 64-bit Window 10 Pro computer running Intel Core i7-2670QM processor with 2.20 GHz and 8 GB RAM. The experiment was repeated 10 times in order to find the average speed of each algorithm.

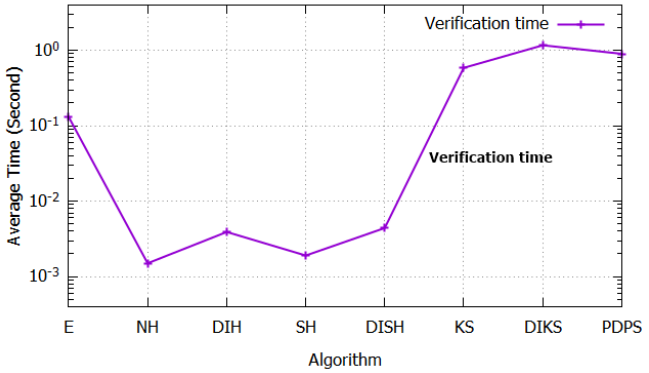

Fig. 7. Verification Time

According to the results shown in Figure 7, In terms of verification time or time taken to successfully login, Hash and Salted Hash functions are the fastest among the other techniques. That is, it usually takes no more than 5 ms to verify a user. Encryption based technique is the second best approach in terms of verification time as it requires roughly 0.1 seconds to login. However, it is vulnerable to password attack in the event of the database leak because the keys used for decrypting password needs to also be stored in the system. Key stretching, on the other hand, takes roughly half a second to verify users because it is specifically designed to prevent brute force and dictionary attacks. The double iteration version of the key stretching technique takes twice as much time, so it is not quite practical in terms of users' experience. In fact, to enhance security in a system using key stretching, one should rather adjust parameters such as thread parallel or amount of allocated memory. Lastly, our proposed PDPS has similar performance to that of Key Stretching technique. In particular, verification time is approximately 0.89 s as shown in the Figure 8.

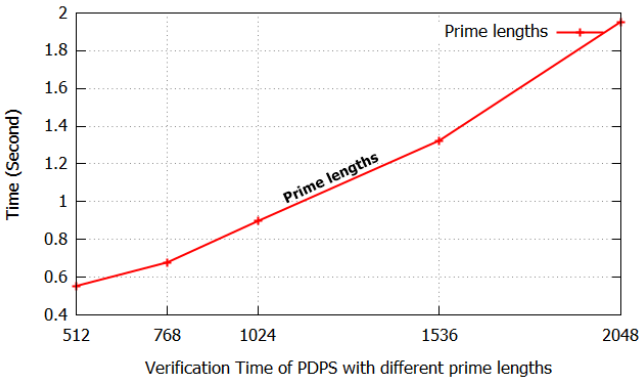

Fig. 8. Verification Time of PDPS with different prime lengths

This implies that users only have to wait for no more than one second to gain access into a system. Hence, this wait time is still acceptable and can be applied to any online applications or services.

B. Storage Space Analysis

In terms of password storage space per user, PDPS stored two values: product of the two random prime numbers (Z) and the prime number that were transformed by the password (Y'). Therefore, the storage space of PDPS is proportional to the prime length. To put this into perspective, we compared the storage space required by PDPS and that required by other existing techniques: Hash Function and Key Stretching. For each technique, we calculated the storage space required by each technique and varied the number of users from 100,000 to 1,000,000. Hash function techniques (md5, sha1, sha3-512) requires the least amount of storage because they only need to store the hash value and some additional salt in the case where salt is being used. Key stretching, on the other hand, required some extra storage space for keeping the adjustable parameters. In the case of argon2, the storage space needed to store the passwords also depends on the length of the hash value which is configurable.

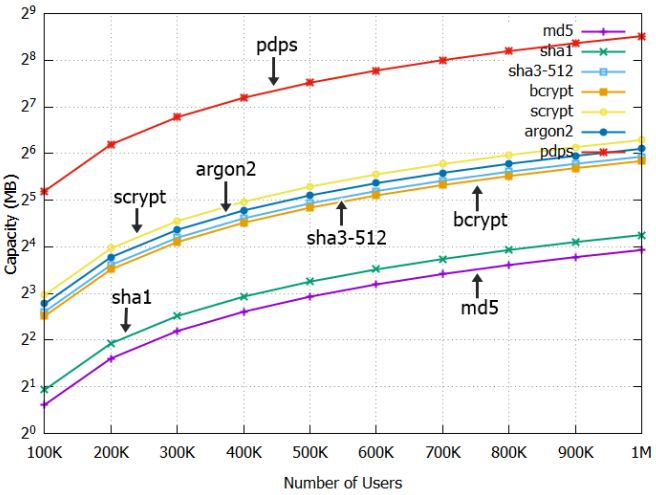

Fig. 9. Storage Capacity Comparison

According to the results shown in Figure 9, our proposed PDPS consumes the largest amount of storage space when compared to the other two main techniques. For one million users, PDPS requires 366.21 MB while MD-5 only requires 15.6 MB. These results were as expected. This is because PDPS is designed to consume space and time in order to discourage hackers from attacking the system, hence, it requires more space than other conventional techniques. While this amount of space may seem to be much larger than the other techniques, it is still considerably small and very much feasible for implementation [28].

C. Security Analysis

In terms of security, we created test data and environment for performing password database attack on various storing techniques. To do this, we select top 1,000 passwords from 10,000 most popular passwords. The passwords were stored in MySQL database using 8 different techniques: Encryption, Normal Hash, Double Iteration Hash, Salted Hash, Double Iteration Hash, Key stretching, Double Iteration Key stretching, and Prime Decomposition.

We used Offline Dictionary Attack Mechanism [34] to test the strength of each technique as shown in Figure 10. This type of attack has been proven to take the least amount of time when compare to Rainbow Table or Brute Force approaches [35], [36].

However, the effectiveness of this type of attack relies solely on the quality of the word list. If any of the passwords stored in the leaked database is in the word list, this password is crackable regardless of how strong the password protection technique is. Therefore, in the case of weak passwords, the strength of the protection algorithm will depend only on the time required to decrypt the password. In other words, the best way that we can protect the password is to increase its complexity and prevent hackers from successfully decrypting the passwords in a given amount of time.

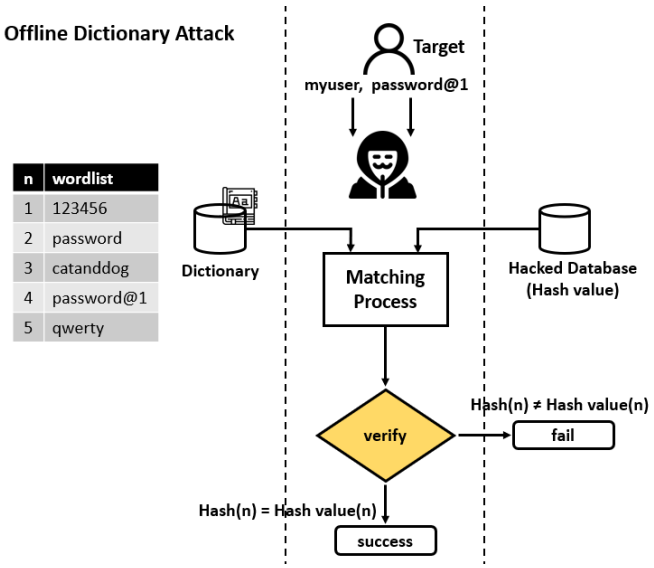

Fig. 10. Offline Dictionary Attack Mechanism

In the experiment, we constructed the word list using the same set of 1,000 passwords mentioned earlier to make sure that the attack will be successful. The size of the dictionary used in the experiments were 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1,000.

We wrote an attack script using PHP and set the maximum execution time value to be infinite in order to continuously

attack every password stored in the database. Cracking time or the time required to crack passwords were recorded and is shown in Table IV.

TABLE IV
CRACKING TIME

Size	Cracking Time (second)							
	E	NH	DIH	SH	DISH	KS	DIKS	PDPS
100	4.47	0.10	0.11	72.06	73.33	693.59	1,415.27	863.44
200	8.70	0.17	0.19	144.29	144.94	1,578.55	2,893.67	1,638.51
300	11.33	0.25	0.29	220.05	221.17	2,245.64	4,153.61	2,560.88
400	15.72	0.32	0.37	279.04	282.55	2,980.12	5,588.64	3,132.41
500	18.96	0.40	0.45	390.99	389.45	3,469.57	7,061.37	3,958.24
600	23.33	0.48	0.52	426.82	427.72	4,158.37	8,120.35	4,933.63
700	26.88	0.53	0.61	547.21	545.70	5,007.11	9,753.20	5,690.22
800	30.14	0.62	0.69	617.98	618.84	5,739.67	11,006.35	6,372.71
900	33.67	0.66	0.78	681.93	680.48	6,374.53	12,537.88	7,185.57
1000	36.00	0.78	0.84	767.67	770.52	7,133.25	13,962.02	7,939.70

According to Table IV, time taken to crack the passwords stored using the hash or salted hash techniques is much less than the time taken by the key stretching techniques and by PDPS. The Normal Hash and the Double Iteration Hash were the weakest among all the techniques considered in the experiment. That is they required the least amount of time and it only took less than 1 second to crack 1,000 passwords. Additional iteration of hash does not increase the security level, as is shown in Figure 11.

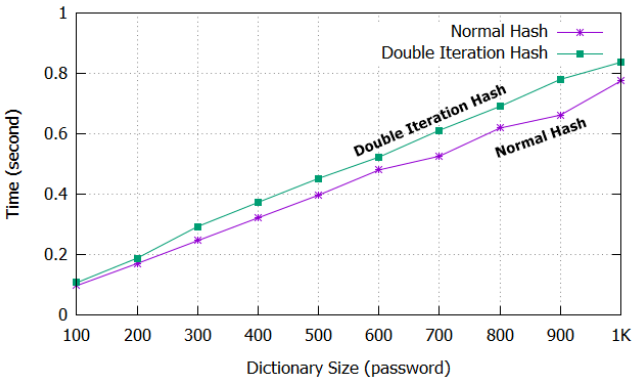

Fig. 11. Hash Cracking Time

By adding salt, the crack time increases over 700 times when compared with the Normal Hash approach. However, doubling the iteration does not give rise to longer crack time. In fact, it took nearly the same amount of time to crack the passwords stored in the leaked database using Salted Hash or Double Iteration Salted Hash techniques, as shown in Figure 12. Finally, time taken to crack the passwords stored by Key stretching and our proposed Prime Decomposition techniques is significantly longer than the other techniques. Cracking 1,000 passwords requires more than 2 hours.

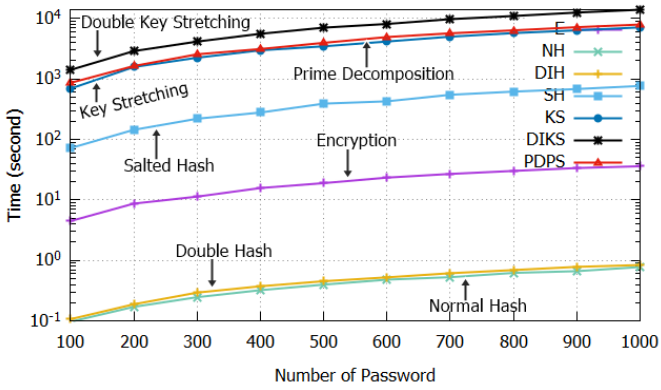

Fig. 12. Cracking Time Comparison

D. Time Complexity Analysis

For time complexity experiment, we assumed that an attacker was able to access the password database which contained all of the information for password verification such as username, hashed password, or any other values used for password verification. In addition, we also assumed that the attacker got the source code of our algorithm [37]. Given the above situation, we find the complexity required to revert the available information back to the plain text passwords.

When the attackers gain access to PDPS password database, he/she will obtain username, the Y' value, and the Z value. Therefore, there are 3 possible scenarios:

1) Brute Force Attack: Gaining access to Z and Y' do not give attackers any additional clue to the original pain text passwords. This is because Y' is a function of the password length and a random number. Therefore, even if the users' passwords are the same, the generated Y' values will always be different if the randomly chosen X values are different. Figure 13 demonstrates how the same password from the same or different users can be stored differently in the database. Hence, the only way to attack is by going through all the different combination of the plain text passwords in a brute force manner to find the one that passes the verification process. This type of attack requires intensive processing power and time especially when the password length is long.

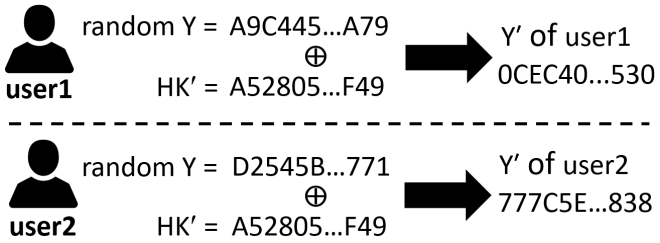

Fig. 13. Example of how the same password from the two different users can get transformed to Y' and stored in the database differently.

2) Dictionary Attack: To reduce the computing power and time required to perform the brute force attack, hackers can use a list of popular passwords to perform a dictionary attack. However, this approach is still time consuming and does not guarantee success: weak passwords are vulnerable to the attacks while strong passwords may be safe from the attacks. To gain more insight into the performance of PDPS, Table VI shows the time complexity required to perform the dictionary

attack on PDPS in comparison to other common password storing techniques [40].

TABLE V
PARAMETERS NOTATION

Symbol	Description
N_d	the number of elements in a password list
N_p	the number of passwords to be cracked
T_h	the time spent on executing a cryptographic hash function
T_{ks}	the time spent on executing a key stretching algorithm
T_{pd}	the time spent on generation surrogate hash key (in our case is the Y' value)
T_{mhash}	the time spent on determining whether two hash values matches
T_{mks}	the time spent on determining whether two passwords enhanced by a key stretching algorithm match
T_{mpd}	the time spent on modulus testing match

TABLE VI
THE COMPARISONS OF ATTACK COMPLEXITY

Methods	Time complexity
Hashed Password	$O(N_d * N_p * (T_h+T_{mhash}))$
Salted Hash Password	$O(N_d * N_p * (T_h+T_{mhash}))$
Key Stretching	$O(N_d * N_p * (T_h+T_{mks}))$
Prime Decomposition*	$O(N_d * N_p * (T_h+T_{pd} + T_{mpd})))$

According to the complexity analysis shown in Table VI, the proposed PDPS is more durable than the hashed password or salted hash password methods. However, we cannot directly compare the complexity of PDPS with the Key Stretching method because complexity of such method is adjustable depending on the parameters used. Hence, complexity of PDPS can either be lower than, equal to, or higher than the Key Stretching approach.

3) Prime Number Factorization Attack: Upon obtaining the Z value, attackers can try to factorize this value into two prime numbers. Factorization will be successful if the attackers can find one of the prime numbers that were used during the password transformation process. The attackers may bypass authentication system and use one of the prime factor (X or Y) instead of using the original password. However, factorizing the Z value is time consuming because the Z value is the product of two large prime numbers that were randomly chosen and never got stored in the database. Due to the difficulty of the large prime number factorization problem, the attackers have to go through all possible values (brute force) to find X or Y. More specifically, time taken to find X or Y is roughly $O(b^k)$ [38] where b is the size of Z (2048 bits) and k is the size of X and Y (1024 bits). Currently, there is no algorithm that can factor large prime number efficiently. Even with the most effective algorithm, General Number Field Sieve (GNFS) [39] which can factor large numbers of more than 100-bit, cannot be used to factor 2048-bit number. Hence, finding plain text password from Z value is nearly impossible, given the state of the current technology.

V. CONCLUSION

This research presents a novel algorithm for securing stored password using prime decomposition techniques. The results show that even if the password database is exposed to public, our proposed PDPS method can securely protect the password because it is specifically designed so that the same password will be transformed to different values before storing in the database. In addition, PDPS does not require any plain text information, such as salt, to be stored in the database. As a result, the only efficient way to crack the password is by using dictionary attack. Thus, PDPS is safe from Time-Memory Attack because it yields slightly higher password verification time than that of the key stretching method.

ACKNOWLEDGMENT

A special thank to the Faculty of Information Technology and Digital Innovation, King Mongkut's University of Technology North Bangkok and Rajamangala University of Technology Phra Nakorn for providing necessary facility and equipment for this research.

REFERENCES

[1] F. E. De Guzman, B. D. Gerardo, and R. P. Medina, "Implementation of Enhanced Secure Hash Algorithm Towards a Secured Web Portal," in *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, Feb. 2019, pp. 189–192.

[2] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Advances in Cryptology – EUROCRYPT 2005*, Berlin, Heidelberg, 2005, pp. 19–35.

[3] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, Mar. 2016, pp. 292–302.

[4] C. Percival, "Stronger key derivation via sequential memory-hard functions", Proc. BSDCan, pp. 1-16, 2009-May.

[5] F. van den Broek and E. Poll, "A Comparison of Time-Memory Trade-Off Attacks on Stream Ciphers," in *Progress in Cryptology – AFRICACRYPT 2013, Berlin, Heidelberg*, 2013, pp. 406–423.

[6] S. Boonkrong and C. Somboonpattanakit, "Dynamic Salt Generation and Placement for Secure Password Storing," *IAENG International Journal of Computer Science*, vol. 43, no. 1, pp27–36, 2016.

[7] B. Harsha and J. Blocki, "Just In Time Hashing," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, Apr. 2018, pp. 368–383.

[8] P. A. Grassi, M. E. Garcia, J. L. Fenton, "Digital Identity Guidelines", standardized by NIST in *NIST Special Publication 800-63-3* and published in June 2017.

[9] S. K. Sood, A. K. Sarje, and K. Singh. "Cryptanalysis of password authentication schemes." *Proceeding of International Conference on Models in Computer Science (ICM2CS 2009)*. pp. 1–7, 2009.

[10] J. Basilio-Ramirez, H. Perez-Meana, and V. Ponomaryov, "Multifactor authentication system based on biometrics and radio frequency identification," in *2016 9th International Kharkiv Symposium on Physics and Engineering of Microwaves, Millimeter and Submillimeter Waves (MSMW)*, 2016, pp. 1–4

[11] S. Janakiraman, K. S. Sree, V. L. Manasa, S. Rajagopalan, K. Thenmozhi, and R. Amirtharajan, "OTP on Demand - An Embedded System for User Authentication," in *2018 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2018, pp. 1–5, doi: 10.1109/ICCCI.2018.8441400.

[12] A. Adukkathayar, G. S. Krishnan, and R. Chinchole, "Secure multifactor authentication using NFC," in *2015 10th International Conference on Computer Science Education (ICCSE)*, 2015, pp. 349–354.

[13] S. Khandelwal, 2016. VK.com HACKED! 100 Million Clear Text Passwords Leaked Online Available online at https://thehackernews.com/2016/06/vk-com-data-breach.html

[14] B. Bhat, A. W. Ali, and A. Gupta, "DES and AES performance evaluation," in *Communication Automation International Conference on Computing*, May 2015, pp. 887–890.

[15] Y. Liu et al., "Design of password encryption model based on AES algorithm," in *2019 IEEE 1st International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*, Oct. 2019, pp. 385–389.

[16] M. A. D. Brogada, A. M. Sison, and R. P. Medina, "Cryptanalysis on the Head and Tail Technique for Hashing Passwords," in *2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)*, Dec. 2019, pp. 137–142.

[17] P. Gauravaram, "Security Analysis of salt——password Hashes," in *2012 International Conference on Advanced Science and Technologies (ACSAT)*, 2012, pp. 25–30

[18] M. Hellman, "A cryptanalytic time-memory trade-off," *IEEE Transactions on Information Theory*, vol. 26, no. 4, pp. 401–406, Jul. 1980

[19] J. Zhou, J. Chen, K. Pan, C. Zhao, and X. Li, "On the security of key derivation functions in office," in *and Identification Anti-counterfeiting*, Security, Aug. 2012, pp. 1–5.

[20] K. Balasubramanian, "Variants of RSA and their cryptanalysis," in *2014 International Conference on Communication and Network Technologies*, Dec. 2014, pp. 145–149.

[21] L. Wu, H. J. Cai, and Z. Gong, "The Integer Factorization Algorithm With Pisano Period," *IEEE Access*, vol. 7, pp. 167250–167259, 2019.

[22] M. Raza, M. Iqbal, M. Sharif and W. Haider "A Survey of Password Attacks and Comparative Analysis on Methods for Authentication" *World Applied Sciences*, 2012, pp. 439-444.

[23] E. R. Andrade, M. A. Simplicio, P. S. L. M. Barreto, and P. C. F. d Santos, "Lyra2: Efficient Password Hashing with High Security," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 3096–3108, Oct. 2016.

[24] A. Arshad, D.-S. Kundi, and A. Aziz, "Compact implementation of SHA3-512 on FPGA," in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, Jun. 2014, pp. 29–33.

[25] Y. Wu and X. Wu, "Implementation of efficient method of RSA key-pair generation algorithm," in *2017 IEEE International Symposium on Consumer Electronics (ISCE)*, Nov. 2017, pp. 72–73.

[26] K. Balasubramanian, "Variants of RSA and their cryptanalysis," in *2014 International Conference on Communication and Network Technologies*, Dec. 2014, pp. 145–149.

[27] OpenSSL Management Committee (OMC), 2019. OpenSSL Strategic Architecture Available online at https://www.openssl.org/docs/OpenSSLStrategicArchitecture.html.

[28] D. Tse, K. Huang, B. Cai, and K. Liang, "Robust Password-keeping System Using Block-chain Technology," in *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Dec. 2018, pp. 1221–1225.

[29] Claude E. Shannon, "A Mathematical Theory of Cryptography," *Bell System Technical Memo MM 45-110-02*, September 1, 1945.

[30] B. Mark, 2014. 10,000 most common passwords list Available online at https://xato.net/passwords/more-top-worst-passwords/.

[31] N. Floissac and Y. L'Hyver, "From AES-128 to AES-192 and AES-256, How to Adapt Differential Fault Analysis Attacks on Key Expansion," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Sep. 2011, pp. 43–53.

[32] A. Arshad, D.-S. Kundi, and A. Aziz, "Compact implementation of SHA3-512 on FPGA," in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, Jun. 2014, pp. 29–33.

[33] X. Wu and S. Li, "High throughput design and implementation of SHA-3 hash algorithm," in *2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, Oct. 2017, pp. 1–2.

[34] L. Bošnjak, J. Sreš, and B. 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)*, May 2018, pp. 1161–1166.

[35] E. Liu, A. Nakanishi, M. Golla, D. Cash, and B. Ur, "Reasoning Analytically about Password-Cracking Software," in *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 380–397.

[36] F. Yu and Y. Huang, "An Overview of Study of Passowrd Cracking," in *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, Oct. 2015, pp. 25–29.

[37] Auguste Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, vol. IX, pp. 5–83, Jan. 1883, pp. 161–191.

[38] Krantz, Steven G., "The Proof is in the Pudding: The Changing Nature of Mathematical Proof," *New York: Springer*, 2011, p. 203.

[39] H. Yu and G. Bai, "An efficient method for integer factorization," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May. 2015, pp. 73–76.

[40] W. Luo, Y. Hu, H. Jiang, and J. Wang, "Authentication by Encrypted Negative Password," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 1, pp. 114–128.

[41] A. K. Tarafder and T. Chakroborty, "A Comparative Analysis of General, Sieve-of-Eratosthenes and Rabin-Miller Approach for Prime Number Generation," in *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, Feb. 2019, pp. 1–4.

[42] Miller, L. Gary, "Riemann's Hypothesis and Tests for Primality," in *1976 Journal of Computer and System Sciences*, vol. 13, no. 3, Dec, 1976, pp. 300–317.

[43] W. T. Penzhorn, "Fast algorithms for the generation of large primes for the RSA cryptosystem," in *1992 Proceedings of the South African Symposium on Communications and Signal Processing*, Sep. 1992, pp. 169–172.

[44] C. Duta, L. Gheorghe, and N. Tapus, "Framework for Evaluation and Comparison of Primality Testing Algorithms," in *2015 20th International Conference on Control Systems and Computer Science*, May. 2015, pp. 483–490.

Chaovalit Somboonpattanakit received his B.Sc. in Information Systems from Rajamangala University of Technology Phra Nakorn, where he also work as a computer and network specialist. He is currently pursuing his Ph.D. in Information Technology at the Faculty of Information Technology and Digital Innovation, King Mongkut's University of Technology North Bangkok (KMUTNB), Thailand. Chaovalit also hold several Cisco CCNA and Microsoft certificates. His research interest is mainly on network security.

Nawaporn Wisitpongphan is an Assistant Professor in the Faculty of Information Technology and Digital Innovation and also a director of the Research Center of Information and Communication Technology at King Mongkut's University of Technology North Bangkok (KMUTNB), Thailand. She received her B.S., M.S., and Ph.D., in Electrical and Computer Engineering from Carnegie Mellon University in 2000, 2002, and 2008, respectively. Prior to joining KMUTNB, she was a researcher at General Motor Research Center, Warren, Michigan. While her expertise is in computer network, vehicle-to-vehicle communication, her current research focus is on social network analysis, information technology management, and Smart environments.