Journal of Emerging Computer Technologies
Research Article
Received: *2021-12-03* | ***Reviewing:*** *2021-12-03* & *2021-12-28* | ***Accepted:*** *2021-12-28* | ***Online:*** *2021-12-28* | ***Issue Date:*** *2021-12-31*

A Dynamic Method and Program for Multiple Password Generation and Management

Onur Çakırgöz
Department of Computer Engineering
Bartın University, Bartın, Turkey
onurcakirgoz@bartin.edu.tr
0000-0002-9347-1105

Süleyman Sevinç
Labenko Bilişim A.Ş.
İzmir, Turkey
suleysevinc@gmail.com
0000-0001-9052-5836

Abstract— **Authentication is a process that should be fulfilled by users to gain access to websites/services. Today, the most common method used for authentication is still text-based passwords. However, several difficulties/problems are encountered during the use of passwords for authentication. One of them is that users must use a separate and strong password for each different website. Unfortunately, rather than using distinct passwords, users generally prefer to use the same password or similar passwords for different services, which inevitably leads to security vulnerabilities. Therefore, there is a need for a method/program that will enable easy and secure management of many strong passwords. In this study, a dynamic method and program is proposed to solve this problem. This method and program, inspired by the Chinese Remainders Theorem (CRT), simplifies the generation and management of multiple passwords. With this program, many individual passwords can be generated from a single unique password. Both the unique password and the individual passwords are not stored anywhere. The only thing users need to remember is the unique password, and in our method, long but easy-to-remember unique passwords can be used safely. Although inspired by the CRT, our method is not based on the CRT. CRT is only used in the security analysis of our method.**

Keywords— password, password management, password generation, chinese remainder theorem, authentication

I. INTRODUCTION

Since text-based password is economical and relatively easy to use, it is still the most widely used authentication method for electronic services [1], [2]. Besides, the number of websites that users visit to receive service, and in parallel, the number of passwords that they need to use are increasing day by day [1]. However, the password usage that increases in daily life has some consequences. The more passwords users have, the harder they are to remember. Also, since non-random and weak passwords are known to be prone to dictionary attacks, companies and institutions define policies/constraints on the password's strength [3]. The definition of some constraints by service providers upon the strength(predictability) of passwords, significantly increases the mental burden of the users [4]. From this point of view, many studies were conducted investigating password habits/behaviors of users [1], [2], [5], [6], [7], [8]. These studies showed that a significant portion of users, even some experts, use very similar passwords for different services, or even the same password. On the other hand, the password setting patterns of the users are also very similar. For example, most of the users use capital letters at the beginning of passwords, and numbers and punctuation at the end of passwords. Thus, particular service provider's security policies that are applied to the user passwords encounter the threat of losing the effects. In the worst-case, when a user uses same password for n different services, a malicious server/person can impersonate him/her and have illegal access to these n services.

Among the approaches that facilitate the use/management of passwords, password managers and password generators are the two prominent approaches. Password managers are applications that store user passwords in a place (locally or in a remote location) and retrieve passwords when requested. Most web browsers we use today also have this feature. Password generators, on the other hand, are applications that generate strong (hard to remember and guess) and random passwords and regenerate them when needed. The basic approach used by the majority of password generators is as follows: Generating new passwords by blending several pieces of information with a mathematical method. From this perspective, the two things that make the difference between password generators are the mathematical method used and the types of information blended. The password generator called *Site-Specific Passwords (SSP)* [9] is probably one of the oldest proposed solutions. The SSP creates a password by combining a user master password and an easy-to-memorize name that the user specifies for the website. The main problem with this approach is that as the number of websites increases, the user is likely to forget the names he/she set (especially for rarely used websites). The password generator named *Password Multiplier* [10] creates a password by combining the user master password, the website name, and the username for the website. *PasswordSitter* [11] generates a password as a function of the following components: user master password, user ID, website/service name, and some configurable parameters. The *Passpet* [12] application generates a password as a function of the user's master password and the name-icon pair selected by the user for the website. *ObPwd* [13] generates a password using two mandatory components (user master password and a website-specific object selected by the user) and one optional component (URL). *PALPAS* [14], unlike previous solutions, uses server-provided password policy data to generate passwords that meet the website's requirements. Password policy data is a small file created with a simple XML-based language. The other two components that *PALPAS* uses to generate the password are a stored master password and a client-side secret that is synchronized across all user devices. On the other hand, *AutoPass* [15] is a solution similar to *PALPAS* in terms of architecture and method. *AutoPass* fixed some of the shortcomings/problems of the previously proposed password generators. The Autopass client, which is an add-on to the web browser, communicates with the Autopass server. In case of Autopass server crash, the user cannot access (generate) passwords. Abderrahim Abdellaoui et al. proposed a scheme that performs authentication in a cloud environment [16]. The proposed scheme is based on a password generator that uses/blends methods such as multifactor authentication, one-time password, and SHA1. Finally, drPass [17] is a protocol that helps users create and maintain different passwords for each web site. For drPass to be used, websites must accept this protocol and change their architecture accordingly.

There are also password generators developed as add-ons to various web browsers or as mobile applications. For example, *RndPhrase* (https://rndphrase.appspot.com) is a Firefox extension and similarly generates passwords as a function of some information. *PwdHash port* (https://addons.opera.com/en-gb/extensions/details/pwdhash-port) is an Opera extension and based on the *PwdHash* [18] password generator. On the other hand, *Password generator* (https://goo.gl/SNVtJY) is an Android application and generates passwords using configurable information.

Password managers are applications that help users to store and manage multiple passwords. Password manager programs are based on the basic principle of storing all user passwords (in encrypted form) in one place (usually locally). The user can access all the passwords stored locally with a unique password called the master key, which the user must know by heart. Among the current password management programs, *LastPass* (https://www.lastpass.com), *KeePass* (https://keepass.info), *Dashlane* (https://www.dashlane.com) and *1Password* (https://1password.com) stand out. When these four password manager programs are examined, it is seen that they all have the ability to automatically generate strong passwords depending on the user's request. From this point of view, these password managers also have a password generator feature. In addition, these four programs encrypt the database holding individual passwords with the AES-256 encryption algorithm. Most current online password managers learn or store account passwords and/or master passwords. In a recent study, this security vulnerability was addressed and *HIPPO* [19], a cloud-based password manager protocol that does not learn or store master passwords and account passwords, was proposed for the design of online password managers.

Single sign-on (SSO) protocols offer the opportunity to automatically share identity/authentication data between different sites [20], [21]. In this approach, a user authenticates only once and thus gains access to different web sites without having to re-authenticate. However, for this, SSO protocols require involved parties to first establish a circle of trust. Unfortunately, each service/site has different security requirements/levels and it's impossible to create a single unique federation where all parties trust each other [22]. Therefore, since there is no single unique SSO federation, the SSO approach is not able to provide a definite solution to the problem of managing the large number of passwords users have [23].

CRT is an ancient theorem which is frequently used in number theory, and it was originated by a Chinese mathematician Sun Tzu. To date, CRT was used in many scientific studies in both mathematics and computer science fields for different purposes. One of the earliest and well-known studies in computer science using the CRT is the (t,n) threshold scheme [24] proposed by Asmuth & Bloom. In 2007, S. Iftene presented a multi-authority e-voting scheme based on CRT [25]. J. C. Patra et al. proposed a novel CRT-based technique for digital watermarking in 2010 [26]. One year later, S. K. Kim et al. proposed new modular exponentiation and CRT recombination algorithms secure against all known power and fault attacks [27]. In 2014, K. Kaya, and A. A. Selçuk proposed a new threshold scheme for the Digital Signature Standard (DSS) using Asmuth–Bloom secret sharing based on the CRT [28]. There are also many recent studies in the literature that use the CRT for different purposes such as secret sharing [29], image sharing [30], key agreement [31], data encryption [32] and data compression [33].

Due to security requirements, users must use a separate and strong (cannot be guessed easily and hard to remember) password for each different website. It is nearly impossible for users to memorize/remember large numbers of strong passwords. Therefore, there is a need for a method/program that will enable easy and secure management of large numbers of passwords. In this study, a dynamic method and program is proposed for this need. The sections of the article are as follows: In the second section, firstly, it is mentioned about the passwords and the conversion of passwords into a numeric value with polynomial representation, then, the CRT is introduced, its formulas are given, and finally, the method and program we developed for the generation and management of multiple passwords are explained. In the third chapter, sample individual passwords produced by our program are shown, and the properties and security of our method is discussed. Final remarks are given in the last section.

II. CHINESE REMAINDER THEOREM

CRT is about finding a solution to the system of simultaneous congruences. Suppose that X, a and p are positive integers. Then, (1) defines a congruence.

$$X \equiv a(mod\ p) \quad (1)$$

A system of simultaneous congruences is defined in (2). Here $p_1,p_2,...,p_n$ should be pairwise coprimes. Namely, greatest common divisor; gcd(p_i, p_j) should be 1 for all $1\leq i,j \leq n$ and $i\neq j$. Then, this system of simultaneous congruences has a unique solution X (mod r).

$$X \equiv a_i\ (mod\ p_i)\ (i = 1,2,\cdots,n) \quad (2)$$

Given,

$$r = \prod_{i=1}^{n} p_i \quad (3)$$

Let,

$$M_i = \prod_{j=1, j\neq i}^{n} p_j\ \ (i = 1,2,\cdots,n) \quad (4)$$

Then, unique solution *X* is computed as in (5):

$$X = (\sum_{i=1}^{n} a_i M_i (M_i^{-1} mod\ p_i))\ (mod\ r) \quad (5)$$

III. THE DYNAMIC METHOD AND PROGRAM FOR MULTIPLE PASSWORD GENERATION AND MANAGEMENT

For the generation and management of multiple passwords, firstly, a method called the backward direction method was developed. In fact, the backward direction method is exactly based on the CRT. Unfortunately, as a result of the evaluations, it was seen that the method contains some problems, and it is not possible to use it. Then, a different method, called forward direction method, was developed. The forward direction method is not based on the CRT. The CRT was only evaluated in the security analysis of this method. Both methods were described in detail below.

A. Backward Direction Method

Based on CRT, we might think (a_1, a_2, ..., a_n) in (2) as n individual passwords (we already have or to be generated). In response to these, prime numbers (p_1, p_2, ..., p_n) that are greater than these numbers can be generated randomly and can be used to acquire individual passwords. The solution of this system of simultaneous congruences would give us the X, namely, the unique password.

Extracting individual passwords from X is straightforward. In this case, k'th individual password can be computed as $X \equiv a_k \pmod{p_k}$.

Then, what we need to obtain individual passwords are the value of X and the corresponding prime numbers. When X and the respective prime number are put together, the desired individual password can be easily acquired. But, having them individually is not sufficient to obtain the password. In the light of this information, we can define the required steps for backward direction method:

1) Convert each of the n passwords to integer by using Horner method.

2) For each password, generate a prime number that is greater than password and distinct from each other.

3) Compute the unique password X from the equation system.

4) Store prime numbers in a file. Also, remove n passwords.

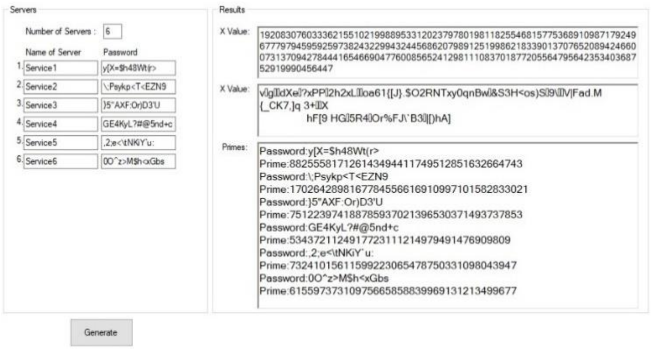
The "Backward_Direction_Method()" algorithm generates unique password X and prime numbers in accordance with the backward direction method:

```
Backward_Direction_Method(string[] passwords)
01  n = passwords.Length
02  Let Num_psw[0..(n+1)] be a new Biginteger array
03  for i = 0 to n-1
04      Num_psw[i] = Convert_String_to_Integer(passwords[i])
05  for j = n to (n+1)
06      number = Randomly generate a Biginteger value
07      Num_psw[j] = number
08  Let primes[0..(n+1)], Mi_values[0..(n+1)] and I_o_Mi[0..(n+1)] be new Biginteger arrays
09  primes = Find_primes_for_passwords(Num_psw, n+2)
10  Mi_values = Find_Mi_Values(primes)
11  M_value = Find_M_Value(primes)
12  I_o_Mi = Return_inverse_of_Mi(Mi_values, primes)
13  X_value = Compute_X(Num_psw, Mi_values, I_o_Mi, primes, M_value)
14  X = Convert_Integer_to_String(X_value)
15  Print X to the screen, store first n primes in a file
```

In order to increase the security of the backward direction method, two extra pairs of passwords and prime numbers are created in the algorithm, and these are included in the solution set. However, extra generated passwords and prime numbers are discarded and are not shown to the user. The reason for generating two extra password-prime number pairs is to prevent a malicious person from finding the unique password by obtaining some or all of the password-prime number pairs of the user.

The details and working logic of the algorithm are as follows: The "passwords" parameter is a string array, and it holds individual passwords. In line 1, the number of individual passwords is assigned to the "n" variable. The "Num_psw" array is defined in Line 2, and this is used to hold numerical equivalents of individual passwords. There is a for loop in Lines 3 and 4, and the numerical equivalents of all individual passwords are assigned to the first n elements of the "Num_psw" array. Similary, there is a for loop between Lines 5 and 7, and randomly generated two biginteger values are assigned to the last 2 elements of the "Num_psw" array. In line 8, "primes", "Mi_values" and "I_o_Mi" arrays are defined, and they hold prime numbers, M_i values in (4), and inverse of M_i values (M_i^{-1}) in (5), respectively. In line 9, prime numbers are generated randomly by the "Find_primes_for_passwords()" method and they are assigned to the "primes" array. In line 10, M_i values are calculated by the "Find_Mi_Values()" method, which corresponds to (4), and then, they are assigned to "Mi_values" array. In line 11, M value (r in (3)) is calculated by the "Find_M_Value()" method, which corresponds to (3), and then, it is assigned to "M_value" variable. In line 12, inverses of M_i values (M_i^{-1}) calculated by the "Return_inverse_of_Mi()" method are assigned to "I_o_Mi" array. In line 13, the integer equivalent of the master password is calculated by the "Compute_X()" method, which corresponds to (5), and then, it is assigned to "X_value" variable. In line 14, the unique (master) password is obtained by converting the integer stored in "X_value" variable to string. Finally, in line 15, the master password is shown on the screen and the first n primes are stored in a file.

The password generation page in backward direction method is shown in Fig. 1. The user can easily generate the master (unique) password by entering individual passwords via the form in Fig. 1.



Number of Servers : 6

Name of Server	Password
1. Service1	y[X=$h48Wt(r>
2. Service2	\;Psykp<T<EZN9
3. Service3	}5"AXF:Or)D3'U
4. Service4	GE4KyL?#@5nd+c
5. Service5	,2;e<\tNKiY'u:
6. Service6	0O^z>M$h<xGbs

```
X Value: 192083076033362155102199889533120237978019811825546815775368910987179249
         677797945959259738243229943244568620798912519986218339013707652089424660
         073137094278444165466904776008565241298111083701877205564795642353403687
         52919990456447

X Value: v▯g▯dXe▯?xPP▯2h2xL▯oa61{[J}.$O2RNTxy0qnBw▯&S3H<os)S▯9▯V|Fad.M
         {_CK7,]q 3+▯X
                   hF[9 HG▯5R4▯Or%FJ\`B3▯|[)hA]

Primes:  Password:y[X=$h48Wt(r>
         Prime:88255581712614349441174951285163266474 3
         Password:\;Psykp<T<EZN9
         Prime:170264289816778455661691099710158283302 1
         Password:}5"AXF:Or)D3'U
         Prime:751223974188785937021396530371493737853
         Password:GE4KyL?#@5nd+c
         Prime:534372112491772311121497949147690980 9
         Password:,2;e<\tNKiY'u:
         Prime:732410156115992230654787503310980439 47
         Password:0O^z>M$h<xGbs
         Prime:615597373109756658588399691312134996 77
```

Fig. 1. Password generation page in backward direction method

On the form, user first specifies the number of servers. After that, equal number of textboxes for the names of the servers and for the passwords become visible on the page. Then, user fills the textboxes. Finally, when user clicks on the Generate button, program runs pre-defined methods, writes the names of the server to a text file and prints out unique password, numerical equivalent of unique password, and prime numbers. Furthermore, numeric values of individual passwords, Mi values, inverse of Mi values and M value are printed out. As seen in the figure, the generated unique password X is not a memorable password.

As mentioned previously, users define either similar passwords or same password for different service providers. Obviously, this is a significant security problem. Since individual passwords in backward direction method are defined by the users in advance, the backward direction method does not yield a solution to this security problem. Furthermore, backward direction method has three important drawbacks:

1) The generated X is a very big integer, and the string equivalent of X is not a memorable password. In addition, there are some white-space or control characters in the unique password produced. (As is known, the first 32 characters in ASCII table are not printable - they are control characters.)

2) New passwords cannot be added to the solution system later. Because, when a new individual password and the corresponding prime number are included in the solution system, the unique password changes automatically.

3) As the number of individual password-prime pairs in the equation system increases, the length of the unique password automatically increases.

Due to these three important problems, the backward direction method cannot be used by users in practice. In order to overcome the second problem of the backward direction method (not being able to add new individual passwords to existing individual passwords later), a random prime number (p_i) can be generated first, and then a new individual password can be obtained with X % p_i operation. (This solution approach, with some differences, already corresponds to the basic logic of the forward direction method, which will be explained in the next section.)

B. Forward Direction Method

Instead of starting from the individual passwords, users can first determine the unique password X, which is sufficiently complex but memorable. Secondly, sufficiently large n numbers (divisors) are generated randomly. Here, n numbers should not be pairwise coprime for security reasons. Then, individual passwords can be obtained via the equation (X mod p_k) (for the k'th service). Accordingly, the steps of the forward direction method can be defined as following:

1) Determine a strong unique password X.

2) Convert X into its numerical equivalent with Horner method.

3) Generate n random and distinct integers (divisors) which are not pairwise coprimes.

4) Perform (X mod p_i) for p_1, p_2, ..., p_n. (Here, p_i representation is used for divisors, as well)

5) Convert the results after modulo operation to their string equivalents. Use the results after conversion as individual passwords, but don't store them somewhere.

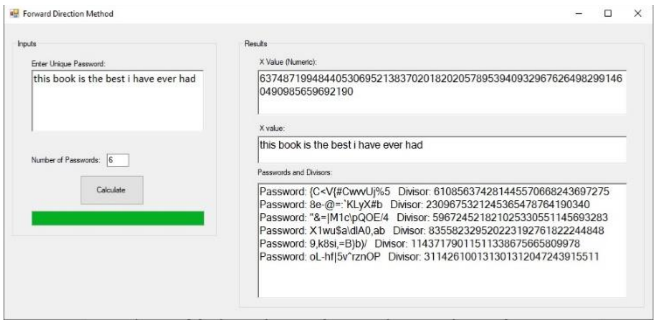
6) Store n divisors in a file.

The "Forward_Direction_Method()" algorithm generates individual passwords and divisors in accordance with the forward direction method. The details and working logic of the algorithm are as follows: The "password" parameter is the unique password. On the other hand, the "number" parameter shows the number of individual passwords (and naturally, the number of divisors) to be generated. In line 1, the integer equivalent of the unique password is calculated, and this value is assigned to the variable "Xvalue". The "divisors" and "passwords" arrays are defined in line 2, and these arrays hold the divisors and individual passwords that will be presented to the user, respectively. There is a while loop between lines 4 and 11, and the two variables ("flag" and "index") defined in line 3 are used appropriately in this loop. In line 5, a divisor is generated randomly by the "Find_divisor_number()" method and this divisor is assigned to the "newdivisor" variable. ("newdivisor" is of type Biginteger.). In line 6, the individual password is obtained by converting the result of the operation (the numeric equivalent of the unique password % the divisor produced) to string, and the individual password is assigned to the variable "psw". In the if statement on the 7th line, it is checked whether the individual password meets the criteria/ restrictions (e.g., minimum-maximum or fixed length, the variety of symbols the password is required to contain, etc.) and whether the currently generated divisor is different from the previously generated ones. Here, since different constraints can be defined for each individual password, the "Control_password()" algorithm checks the individual password with password-specific constraints. Also, "Control_divisor()" algorithm in line 7 checks whether the currently generated divisor and the previously generated ones are pairwise coprimes. If the individual password and the divisor meet the criteria, they are recorded in the respective arrays. If the specified number of individual passwords are generated, the loop is terminated (Lines 11-12). Finally, in line 13, all divisors are stored in a file.

```
Forward_Direction_Method(string password , int number)
01  Xvalue = Convert_String_to_Integer(password)
02  Let divisors[0..number-1] be a new Biginteger array and
    passwords[0..number-1] be a new string array
03  flag = true, index = 0
04  while(flag)
05      newdivisor = Find_divisor_number()
06      psw = Convert_Integer_to_String(Xvalue %
        newdivisor)
07      if (Control_password(psw) &&
        Control_divisor(divisors, newdivisor))
08          divisors[index] = newdivisor
09          passwords[index] = psw
10      index = index + 1
11      if (index = = number)
12          flag = false
13  Store divisors in a file
```

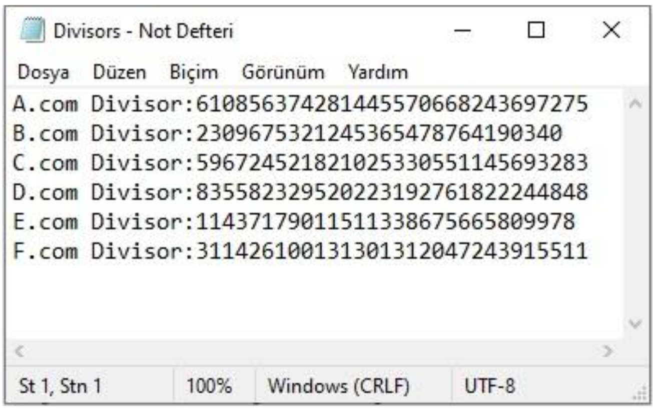
The password generation page in forward direction method is shown in Fig. 2. The user can easily generate individual passwords by entering the unique password via the form in Fig. 2. The details are as follows: If the user clicks on the Calculate button after entering the unique password and the number of passwords to be generated, the program asks for the properties (restrictions) of each individual password. After this stage, the program executes predefined methods and shows the generated passwords and divisors on the screen. As a general constraint, the generated individual passwords consist of only keyboard characters. This is provided via a simple function ("Control_password()") which checks the generated passwords. If the numerical equivalents of the characters constituting a password is between [33..125] in the ASCII table, this string is accepted as a password.

The divisors required to obtain individual passwords are stored in a simple text file together with the server names. An example text file generated by the program and that holds divisors is seen in Fig. 3. In this way, all individual passwords can be obtained easily by entering the unique password to the program. At this point, the operations of the program are quite simple: reading the file that holds the divisors and performing the modulo operations.



```
Forward Direction Method

Inputs
Enter Unique Password:
this book is the best i have ever had

Number of Passwords: 6
Calculate

Results
X Value (Numeric):
637487199484405306952138370201820205789539409329676264982991460490985659692190

X value:
this book is the best i have ever had

Passwords and Divisors:
Password: {C<V{#CwvvUj%5   Divisor: 610856374281445570668243697275
Password: 8e-@=:`KLyX#b   Divisor: 2309675321245365478764190340
Password: "&=|M1c\pQOE/4   Divisor: 596724521821025330551145693283
Password: X1wu$a\dlA0,ab   Divisor: 835582329520223192761822244848
Password: 9,k8si,=B)b)/   Divisor: 11437179011511338675665809978
Password: oL-hf|5v^rznOP   Divisor: 311426100131301312047243915511
```

Fig. 2. Password generation page in forward direction method



```
Divisors - Not Defteri
Dosya  Düzen  Biçim  Görünüm  Yardım
A.com Divisor:610856374281445570668243697275
B.com Divisor:2309675321245365478764190340
C.com Divisor:596724521821025330551145693283
D.com Divisor:835582329520223192761822244848
E.com Divisor:11437179011511338675665809978
F.com Divisor:311426100131301312047243915511

St 1, Stn 1   100%   Windows (CRLF)   UTF-8
```

Fig. 3. The text file generated by the program and that holds divisors

IV. RESULTS AND DISCUSSION

The number of all possible passwords that can be generated in the forward direction method and the properties of the passwords depend on many parameters. These parameters can be listed as follows:

- Size of symbol space
- Unique password defined by the user
- Minimum character length constraint defined on the unique password
- Divisors randomly generated by the program
- Minimum value constraint defined on divisors
- Various constraints on the individual passwords to be generated (minimum character length, the variety of symbols the password is required to contain)

On the other hand, in order to provide the minimum character length constraint defined on individual passwords, randomly generated divisors must also be larger than the numerical equivalents of individual passwords. The parameter values determined in the developed application and the constraints defined for passwords are given in Table-1. Based on these parameter values and defined constraints, nearly 93^{13} different individual passwords which consist of 13 characters can be generated. The relevant constraints given in Table-1 were applied to all individual passwords to be generated. That is, the same constraints were defined for all individual passwords. In accordance with the parameter values and constraints specified in Table-1, sample individual passwords, divisor numbers and numerical equivalents of passwords produced by the application are given in Table-2. In Table-2, the divisor numbers used to generate individual passwords range from 28 to 30 digits. Similarly, the numerical equivalents of individual passwords range from 27 to 30 digits. On the other hand, in this example, "this book is the best i have ever had" is chosen as the master password. This is a very easy phrase to memorize and remember.

TABLE I. PARAMETER VALUES IN THE APPLICATION AND CONSTRAINTS DEFINED FOR PASSWORDS

Parameter/Constraint	Value
The Length of Symbol Space(s)	93 (keyboard characters whose decimal values are between 33 and 125 in the ASCII table)
The minimum Length of Unique Password	30
The minimum Length of Individual Passwords	13
Other Constraints Defined for Individual Passwords	Must contain at least one uppercase letter, at least one lowercase letter, at least one number, and at least one punctuation mark.

TABLE II. SAMPLE INDIVIDUAL PASSWORDS GENERATED IN THE FORWARD DIRECTION METHOD AND OTHER ASSOCIATED DATA

Property	Value
Unique Password	this book is the best i have ever had
Numeric Equivalent of Unique Password	637487199484405306952138370201820205789539409329676264982991460490985659692190 (78 digits)
Number of Individual Passwords Generated	6
The Generated Individual Password and Divisor Pairs	Password1: {C<V{#CwvvUj%5 Numeric_value_of_Password1: 276201135629902806667468828515 (30 digits) Divisor1: 610856374281445570668243697275 (30 digits) Password2: 8e-@=:`KLyX#b Numeric_value_of_Password2: 999950350888928822060524610 (27 digits) Divisor2: 2309675321245365478764190340 (28 digits) Password3: "&=|M1c\pQOE/4 Numeric_value_of_Password3: 76697494620780536421904775292 (29 digits) Divisor3: 596724521821025330551145693283 (30 digits) Password4: X1wu$a\dlA0,ab Numeric_value_of_Password4: 197636672696185503159209650158 (30 digits) Divisor4: 835582329520223192761822244848 (30 digits) Password5: 9,k8si,=B)b)/ Numeric_value_of_Password5: 1009721696862893015829571002 (28 digits) Divisor5: 11437179011511338675665809978 (29 digits) Password6: oL-hf|5v^rznOP Numeric_value_of_Password6: 249526971213973380176697533304 (30 digits) Divisor6: 311426100131301312047243915511 (30 digits)

The CRT corresponds to the backward direction method, but the forward direction method is a different approach from the CRT. Our multiple password generation and management program is based on the forward direction method. One of the strong features of the method and the program is that both the unique password and individual passwords are not stored anywhere. Only the divisors needed to obtain individual passwords are kept in a file. The divisors do not mean anything on their own. Moreover, in the forward direction method, even the divisor-individual password pairs are not sufficient to find the unique password. Because, as the CRT states, all p_i's in (1) should be pairwise coprimes. However, in the forward direction method, the divisors are generated in such a way that they are not pairwise coprimes. Therefore, even though we have all the divisor-individual password pairs, it is not possible to find the unique password using the CRT.

As shown in Table-2, there is a huge difference (in terms of numerical size) between the numerical equivalent of the unique password and the divisors. In this example, the numerical equivalent of the unique password is 78 digits, whereas the largest divisor is 30 digits. Another meaning of this is that the quotient values are much larger than the divisor numbers during modulo operations. With a simple mathematical calculation, it can be found that the quotient values correspond to a number of at least 48 digits. On the other hand, today, an ordinary personal computer at 3 GHz can perform approximately 2.7×10^{14} transactions (for a single core of the processor) in 1 day. Even if the individual password and divisor pair are known, the time required to find the unique password using a brute force approach is nearly 10^{34} days.

V. CONCLUSION

In general, the number of websites where users register to get services is quite high, and this number is increasing day by day. Naturally, for security reasons, users must define and maintain strong and distinct passwords for these different websites. On the other hand, memorizing and bearing many strong passwords in mind is a difficult task. From this point of view, in this study, a dynamic method and program is proposed for the generation and management of many distinct and strong passwords. The proposed method is mainly based on modulo operation. In this method, the user first defines an easy-to-remember and relatively long unique (master) password. Then, the user can easily generate individual passwords by entering the program the unique password and the number of individual passwords he/she wants to generate. One of the most important features of our method is that the generated individual passwords are not stored anywhere. Only the divisors used to obtain individual passwords are stored in a simple text file. On the other hand, the characteristics of the generated individual passwords depend on many parameters. Individual passwords with desired properties can be generated by defining different constraints.

Since the equation system created by our method is similar to the CRT, the CRT was evaluated in the security analysis part of the method. Our approach does not introduce new risks for authentication. Without any loss of security, this approach has the potential to increase the usability of password-based authentication, and individual passwords created would not be prone to dictionary attacks. As a result, our multiple password generation and management program provides managing many individual passwords through a master password and makes password-based authentication more practical and easier to manage for users.

REFERENCES

[1] W. A. S. A. Alothman, "Evaluating Passwords User Behavior and the Psychology of Password Management", *International Journal of Engineering and Computer Science*, 8(04), 24586–24602, 2019.

[2] E. Stobert, R. Biddle, "The password life cycle", *ACM Transactions on Privacy and Security (TOPS)*, *21*(3), 1-32, 2018.

[3] P. Arias-Cabarcos, et. al., "Comparing password management software: toward usable and secure enterprise authentication", *IT Professional*, 18(5), 34-40, 2016.

[4] B. Brumen, "System-Assigned Passwords: The Disadvantages of the Strict Password Management Policies", *Informatica*, *31*(3), 459-479, 2020.

[5] Y. Y. Choong, "A cognitive-behavioral framework of user password management lifecycle", In International Conference on Human Aspects of Information Security, Privacy, and Trust, Springer, Cham, 127-137, June 2014.

[6] E. Stobert, R. Biddle, "Expert password management", In International Conference on Passwords, Springer, Cham, 3-20, December 2015.

[7] B. E. Ur, Supporting password-security decisions with data, PhD Thesis, Carnegie Mellon University, 2016.

[8] C. Shen, et. al., "User practice in password security: An empirical study of real-life passwords in the wild", *Computers & Security*, 61, 130-141, 2016.

[9] A. H. Karp, Site-specific passwords, HP Laboratories, Palo Alto, Tech. Rep., May 2003.

[10] J. A. Halderman, B. Waters, E. W. Felten, "A convenient method for securely managing passwords", In Proceedings of the 14th international conference on World Wide Web, 471-479, May 2005.

[11] R. Wolf, M. Schneider, The passwordsitter, Fraunhofer Institute for Secure Information Technology (SIT), Tech. Rep., May 2006.

[12] K. P. Yee, K. Sitaker, "Passpet: Convenient password management and phishing protection", In Proceedings of the second symposium on Usable privacy and security, 32-43, July 2006.

[13] M. Mannan, P. C. van Oorschot, "Passwords for both mobile and desktop computers: ObPwd for Firefox and Android", *USENIX ;login*, 37(4), 28–37, 2012.

[14] M. Horsch, A. Hülsing, J. A. Buchmann, "PALPAS — passwordless password synchronization", In 2015 10th International Conference on Availability, Reliability and Security, 30-39, August 2015.

[15] F. Al Maqbali, C. J. Mitchell, "AutoPass: An automatic password generator", In *2017 International Carnahan Conference on Security Technology (ICCST)*, 1-6, IEEE, October 2017.

[16] A. Abdellaoui, Y. I. Khamlichi, H. Chaoui, "A novel strong password generator for improving cloud authentication", *Procedia Computer Science*, *85*, 293-300, 2016.

[17] S. Panda, S. Mondal, "drPass: A Dynamic and Reusable Password Generator Protocol", In International Conference on Information Systems Security, 407-426, Springer, Cham, December 2018.

[18] B. Ross, et. al., "Stronger Password Authentication Using Browser Extensions", In USENIX Security Symposium, 17-32, August 2005.

[19] M. Shirvanian, et. al., "A hidden-password online password manager", In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 1683-1686, March 2021.

[20] P. Arias-Cabarcos et al., "Blended Identity: Pervasive IdM for Continuous Authentication", *IEEE Security & Privacy*, 13(3), 32–39, 2015.

[21] F. Alaca, P. C. V. Oorschot, "Comparative analysis and framework evaluating web single sign-on systems", *ACM Computing Surveys (CSUR)*, *53*(5), 1-34, 2020.

[22] P. Arias-Cabarcos, et. al., "Comparing password management software: toward usable and secure enterprise authentication", *IT Professional*, *18*(5), 34-40, 2016.

[23] N. Katuk, et. al., "Can single sign-on improve password management? A focus group study", Pattern Analysis, Intelligent Security and the Internet of Things, Advances in Intelligent Systems and Computing, 85-93, Springer, Cham, 2015.

[24] C. Asmuth, J. Bloom, "A modular approach to key safeguarding", *IEEE transactions on information theory*, *29*(2), 208-210, 1983.

[25] S. Iftene, “General Secret Sharing Based on the Chinese Remainder Theorem with Applications in E-Voting”, *Electronic Notes in Theoretical Computer Science (ENTCS),* 186, 67–84, 2007.

[26] J. C. Patra, A. Karthik, C. Bornand, “A novel CRT-based watermarking technique for authentication of multimedia contents”, *Digital Signal Processing,* 20, 442-453, 2010.

[27] S. K. Kim, et. al., “An efficient CRT-RSA algorithm secure against power and fault attacks”, *The Journal of Systems and Software,* 84(10), 1660-1669, 2011.

[28] K. Kaya, A. A. Selçuk, “Sharing DSS by the Chinese Remainder Theorem”, *Journal of Computational and Applied Mathematics,* 259, 495-502, 2014.

[29] Y. H. Chou, G. J. Zeng, X. Y. Chen, S. Y. Kuo, “Multiparty weighted threshold quantum secret sharing based on the Chinese remainder theorem to share quantum information”, *Scientific Reports*, *11*(1), 1-10, 2021.

[30] K. Meng, F. Miao, Y. Xiong, C. C. Chang, “A reversible extended secret image sharing scheme based on Chinese remainder theorem”, *Signal Processing: Image Communication*, *95*, 116221, 2021.

[31] Y. Jiang, Y. Shen, Q. Zhu, “A Lightweight Key Agreement Protocol Based on Chinese Remainder Theorem and ECDH for Smart Homes”, *Sensors*, *20*(5), 1357, 2020.

[32] T. Duseja, M. Deshmukh, “Image compression and encryption using chinese remainder theorem”, *Multimedia Tools and Applications, 78*(12), 16727-16753, 2019.

[33] R. Vidhya, M. Brindha, “Evaluation and performance analysis of Chinese remainder theorem and its application to lossless image compression”, *Journal of Ambient Intelligence and Humanized Computing*, 1-16, 2021.