Virtual Password: a New Way of Protecting Users from Password Theft

Ming Lei†, Yang Xiao†, Chung-Chih Li*, Susan V. Vrbsky†, and Li Liu†

†Department of Computer Science, The University of Alabama,
‡School of Information Technology, Illinois State University
{mlei, yangxiao, vrbsky, lliu}@cs.ua.edu, cli2@ilstu.edu

Abstract—People enjoy the convenience of on-line services, but online environments may bring many risks. In this paper, we discuss how to prevent users’ passwords from being stolen by adversaries. We propose a virtual password concept involving a small amount of human computing to secure users’ passwords in on-line environments. We propose differentiated security mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security. The tradeoff is that the stronger the scheme, the more complex the scheme may be. Among the schemes, we have a default method (i.e., traditional password scheme), system recommended function, user-specified function, user-specified program, etc. A function/program is used to implement the virtual password concept with a trade off of security for complexity requiring a small amount of human computing. We further propose several functions to serve as system recommended functions and provide a security analysis. We analyze how the proposed schemes defend against phishing, key logger, shoulder-surfing attacks, and multiple attacks. For user-specified functions, we adopt secret little functions, in which security is enhanced by hiding secret functions/algorithms.

I. INTRODUCTION

Today, the Internet has melted into our daily lives with more and more services being moved on-line. Besides reading news, searching for information, and other risk-free activities on line, we are also accustomed to other risk-related work, such as paying using credit cards, checking/composing emails, on-line banking, etc. While we enjoy the convenience, we are putting ourselves at risk. Most current commercial websites will ask their users to input their user identifications (IDs) and corresponding passwords for authentication. Once a user’s ID and the corresponding password are stolen by an adversary, the adversary can do anything with the victim’s account, leading to a disaster for the victim. As a consequence of increasing concerns over such risks, how to protect users’ passwords on the web is becoming more and more critical.

Fig. 1 Phishing report received Sept.’05 – Sept.’06 [3]

The secure protocol SSL/TLS [1] for transmitting private data over the web is well-known in academic research, but most current commercial websites still rely on the relatively weak protection mechanism of user authentications via plaintext password and user ID. Meanwhile, even though a password can be transferred via a secure channel, this authentication approach is still vulnerable to attacks as follows.

• Phishing: Phishers attempt to fraudulently acquire sensitive information, such as passwords and credit card details, by masquerading as a trustworthy person or business in an electronic communication [2]. For example, a phisher can set up a fake website and then send some emails to potential victims to persuade them to access the fake website. This way, the phisher can easily get a clear-text of the victim’s password. Phishing attacks are proven very effective. Fig. 1 illustrates the total phishing reports received from September 2005 to September 2006 [3] according to the Anti-Phishing Working Group

• Password Stealing Trojan: This is a program that contains or installs malicious code. There are many such Trojan codes that have been found online today, so here we just briefly introduce two types of them. Key loggers capture keystrokes and store them somewhere in the machine, or send them back to the adversary. Once a key logger program is activated, it provides the adversary with any strings of texts that a person might enter online, consequently placing personal data and online account information at risk. Trojan Redirector was designed to redirect end-users network traffic to a location to where it was not intended [5]. This includes crime ware that changes hosts files and other DNS
specific information, crime ware browser-helper objects that redirect users to fraudulent sites, and crime ware that may install a network level driver or filter to redirect users to fraudulent locations.

- **Shoulder Surfing**: Shoulder surfing is a well-known method of stealing other’s passwords and other sensitive personal information by looking over victims’ shoulders while they are sitting in front of terminals [12]. This attack is most likely to occur in insecure and crowded public environments, such as an Internet Café, shopping mall, airport, etc. [16, 20]. It is possible for an attacker to use a hidden camera to record all keyboard actions of a user. Video of the user’s actions on a keyboard can be studied later to figure out a user’s password and ID.

Many schemes, protocols, and software have been designed to prevent users from some specified attacks. However, to the best of our knowledge, so far, there is not a scheme which can defend against all the types of attacks listed above at the same time.

In this paper, we present a password protection scheme that involves a small amount human computing in an Internet-based environment, which will be resistant to a phishing scam, a Trojan horse, and shoulder-surfing attacks. We propose a virtual password concept involving a small amount of human computing to secure users’ passwords in on-line environments. We propose differentiated security mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security. The tradeoff is that the stronger the scheme, the more complex the scheme is. Among the schemes, we have a default method (i.e., traditional password scheme), system recommended function, user-specified function, user-specified program, etc. A function/program is used to implement the virtual password concept by trading off security for complexity requiring a small amount of human computing. We further propose several functions serving as system recommended functions and provide a security analysis. We analyze how the proposed schemes defend against phishing, key logger, shoulder-surfing attacks, and multiple attacks. In user-specified functions, we adopt secret little functions, in which security is enhanced by hiding secret functions/algorithms. To the best of our knowledge, our virtual password mechanism is the first one which is able to defend against all three attacks together. We further propose a scheme to adopt μTESLA to be used for re-keying and to defend against Phishing.

The proposed functions include a bijective function and two other schemes called codebook and reference functions. Our objective is to produce a function achieving both 1) ease of computation and 2) security. However, since simplicity and security conflict each other, it is a challenging task to achieve both if possible.

The idea of this paper is to add some complexity, through user computations performed by heart/hand or by computation devices, to prevent the three kinds of attacks. There is a tradeoff of how complex the computation by the users can be. One goal is to find an easy to compute but secure scheme for computing.

We believe that for some sensitive accounts such as on-line bank accounts and on-line credit card accounts, users are likely to choose a little additional complexity requiring some degree of human computing in order to make the account more secure.

The rest of the paper is organized as follows. We describe related work about password protection in Section II. In Section III, we propose the idea of the virtual password. In Section IV, we propose differentiated security mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security. In Section V, we propose user-specified functions or programs, in which we propose the concept of secret little functions. We discuss virtual function with a helper-application in Section VI. We discuss virtual functions without a helper-application in general in Section VII, and we propose a randomized linear generation function in Section VIII. Two functions (the codebook approach and reference function approach) are proposed in Section IX. We share our lessons learned in designing virtual functions in Section X. We present μTESLA authentication in Section XI. In Section XII, we describe implementation issues of our scheme and evaluate usability based on our implementation. Finally, we conclude our paper and describe our future work in Section XIII.

II. RELATED WORK

How to shield users’ passwords from being stolen by adversaries is not a new topic, but it is always a hot topic due to the fact that adversaries keep inventing more and more advanced attacks to break current defense schemes. This results in more research on protecting users from such attacks. In this section, we briefly introduce the previous work on defending against user password-stealing attacks for the three major categories.

Phishing attacks are relatively new but very effective. There are two typical types of Phishing. First, to prevent Phishing emails [27, 29, 30], a statistical machine learning technology is used to filter the likely Phishing emails; however, such a content filter doesn’t work correctly all the time. Blacklists of spamming/phishing mail servers are built in [31, 32]; however, these servers are not useful when an attacker hijacks a virus-infected PC. In [11, 24, 25], a path-based verification has been introduced. In [14], a key distribution architecture and a particular identity-based digital signature scheme have been proposed to make email trustworthy. Secondly, to defend against Phishing websites, the authors in [21, 33] have developed some web browser toolbars to inform a user of the reputation and origin of the websites which they are currently visiting. In [6, 7, 8, 9, 10], the authors implement password hashing with a salt as an extension of the web browser [6, 9, 10], a web proxy[13], or a stand alone Java Applet [15]. Regardless of the potential challenges considered in an implementation, such password hashing technology has a roaming problem because not every web browser installs such an extension or sets the web proxy. Another more important challenge is that more and more web browsers need to be designed in which designers
are not reluctant to include specified extensions for each other.

Unlike Phishing, malicious Trojan horses, such as a key logger, are not attacks, and sophisticated users can avoid them. Such programs are also easy to develop [17] and there is a great deal of freeware that you can download from the internet to prevent them. You maybe be advised to install some anti-spyware or anti-virus software package on your machine, or set up some firewall to block those suspicious packages from the outside. However, in the event you are traveling to some place without carrying your own computer, and you have to seek help from an Internet café to access the Internet, do you want to trust the computers in the internet café? In [17], the author finds a tricky method which can confuse a key logger, which works as follows. Instead of typing your whole password into the login field, the user changes focus outside the login form and types some random characters between any two successive password characters. However, this trick doesn’t shield the user from the keylogger attack, but makes it a little more difficult for adversaries because it is very easy to record all the keys, mouse events, and applications of the focus. The authors in [18, 19] use a virtual pad for the login system, which allows a user to click the virtual keyboard on screen instead of typing in the physical keyboard, but such a virtual key board faces some of the same problems as above, i.e., an adversary can record all the mouse events with a combination of screen snapshots to figure out what the user clicks on the screen.

Alphanumeric password systems are easily attacked by shoulder-surfing, in which an adversary can watch over the user’s shoulder or record the user motions by a hidden camera when the user types in the password. In [22], the authors adopt a game-like graphical method of authentication to combat shoulder-surfing; it requires the user to pick out the passwords from hundreds of pictures, and then complete rounds of mouse-clicking in the Convex Hull. However, the whole process needs the help of a mouse and it takes a long time. In [23], the authors propose a scheme to ask a user to answer multiple questions for each digit. In this way, it is resistant to shoulder-surfing only to a limited degree, because if an adversary catches all the questions, then they will know what the password is. In [23], a game-based method is designed to use cognitive trapdoor games to achieve a shield for shoulder-surfing. The author in [26] has filed a patent to allow a user to make some calculations based on a system generated function and random number for the user to prevent password leaking. However, the scheme in [26] is not anti-Phishing and the password can possibly be stolen if an adversary uses a camera to record all the screens of the system and motions of the victim.

In the above, we have briefly introduced some schemes and did not include methods needing hardware support. Any of the schemes above cannot prevent against Phishing, Trojan horse, and shoulder-surfing at the same time.

III. VIRTUAL PASSWORD

To authenticate a user, a system (S) needs to verify a user (U) via the user’s password (P) which the user provides. In this procedure, S authenticates U by using U and P, which is denoted as: S \( \rightarrow \) U: U, P. All of S, U, and P are fixed. It is very reasonable that a password should be constant for the purpose of easily remembering it. However, the price of easily remembering is that the password can be stolen by others and then used to access the victim’s account. At the same time, we can not put P in a randomly variant form, which will make it impossible for a user to remember the password. To confront such a challenge, we propose a scheme using a new concept of virtual password.

A virtual password is a password which cannot be applied directly but instead generates a dynamic password which is submitted to the server for authentication. A virtual password P is composed of two parts, a fixed alphanumeric F and a function B from the domain \( \psi \) to \( \psi \), where the \( \psi \) is the letter space which can be used as passwords. We have \( P=(F, B) \) and \( B(F, R) = P_r \), where R is a random number provided by the server (called the random salt and prompted in the login screen by the server) and \( P_r \) is a dynamic password used for authentication. Since we call \( P=(F, B) \) a virtual password, we call B a virtual function. The user input includes \( (ID, P_r) \), where ID is user ID. On the server side, the server can also calculate \( P_r \) in the same way to compare it with the submitted password.

It is easy for the server to verify the user, if B is a bijective function. If B is not a bijective function, it is also possible to allow the server to verify the user as follows. The server can first find the user’s record from the database based on the user’ ID, and compute \( P_r \), and compare it with the one provided by the user. A bijective function makes it easier for the system to use the reverse function to deduct F’s virtual password.

The user should be free to pick the fixed part of the virtual password. We propose a differentiated security mechanism in the next section to allow the user to choose the virtual function.

IV. PROVIDING DIFFERENTIATED SECURITY THROUGH A VIRTUAL FUNCTION

We have introduced the concept of the virtual password, and next, we detail how to apply it in an internet-based environment. We propose a differentiated security mechanism for system registration, in which the system allows users to choose a registration scheme ranging from the most simple one (default) to a relatively complex one, where a registration scheme includes a way of choosing a virtual function. The more complex the registration, the more secure the system is, and the more user involvement is required. A screenshot of the first step of the proposed registration is shown in Fig.2. No matter whether a virtual function is used not, the user is required to input the read password and ID in Step 2 of the registration.

In Fig. 2, a user has the freedom to choose a default approach in the traditional way, or a more complex scheme as proposed in this paper. A user can choose a recommended virtual function, define his/her own virtual function, or even define a common program to share between the user and the server to calculate the password.
The system recommended approach is that after the system receives a registration request, it automatically generates a function. The users do not have to provide extra information about the function to the server except for some necessary parameters.

The user specified function approach is one in which users themselves can choose any function they like. However, such freedom is based on the assumption that the user has some basic knowledge about virtual functions, which can be introduced by an on-line introduction.

The indirectly-specified approach is that instead of letting either the users or server make the full decision, this approach allows a user to specify the desired security degree, and then the server will assign a function.

An extreme scheme is that the user can even provide a program in C or Java instead of a function. This requires the user to be a very advanced user.

<table>
<thead>
<tr>
<th>Please choose your PIN registration approach among the followings</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ) Default: do not use a virtual function;</td>
</tr>
<tr>
<td>( ) Use a recommended virtual function</td>
</tr>
<tr>
<td>( ) Use function B=XXX.</td>
</tr>
<tr>
<td>( ) Use function B=YYY.</td>
</tr>
<tr>
<td>( ) Use function B=ZZZ.</td>
</tr>
<tr>
<td>( ) Use a user defined function (Note that user and server share a common function specified by the user):</td>
</tr>
<tr>
<td>( ) Indirect-specified system function, please choose a security degree: Low ( ), Medium ( ), High ( ), or Very High ( )</td>
</tr>
<tr>
<td>( ) Use a user defined program (in C or Java) (Note that user and server share a common program specified by the user):</td>
</tr>
</tbody>
</table>

Note that except for the default approach, either human computing is involved or a handheld device which can be programmed to compute the dynamic password is needed.

We could develop a smart application to make the complex calculation for the user, which can run at the mobile device, such as a cellular phone, PDA, personal computer, or programmable calculator, to relieve the user from the complicated calculations and to overcome any short-term memory problem. If such a helper-application is involved, we should make sure that the helper-application itself should be unique to each user account and only work for the corresponding user account.

Regardless of the approach chosen, a user’s registration in the system is similar, i.e., the user submits a user ID and password. The one difference from a traditional approach is that in the virtual password scheme, there is a virtual function, which is a must, to be set during the registration phase.

The server then delivers this function information to the user via some channels, such as, displaying it on the screen or email. The user needs to remember this function together with the password they have chosen or save them in disks or emails. The user-specified password and the system-generated function are combined into a virtual password.

We also notice that a small amount of human-computing is involved in the authentication process. We have to choose $B$ to make the calculation as simple as possible if the helper-application is not used. A user has to remember both the fixed part and the function part, and as a result will require a little bit more effort to remember. However, the virtual password will be resistant to a dictionary attack, which is mostly caused by the fact that users like to create a password which is either related to their own name, date of birth, other simple words, etc.

In a traditional password scheme, users can change their password, and this is also true in our virtual password scheme. Different from the traditional scheme, users can change the fixed part of the virtual password or the virtual function, or even both.

V. USER-SPECIFIED FUNCTIONS OR PROGRAMS

The strongest security approaches are to let the user define a user-specified function or program. Since the chosen function is only known between the server and the user, and the key space of functions are infinite with high-order, these approaches are very secure, even for some simple functions.

In many classical ciphers, secret encryption algorithms (i.e., algorithms kept as secrets) are common. In modern ciphers, encryption algorithms are open to the public but keys of these algorithms are kept as secrets. One reason that modern ciphers seldom choose secret encryption algorithms is that secret encryption algorithms prevent communications among parties such as commercial products, networking protocols, etc. Therefore, the approach that only keys are kept as secrets (small data) and algorithms (large programs) are open to the public for implementation is very popular to modern ciphers.

The reason that we have a choice of using secret encryption algorithms (i.e., user-specified virtual functions) is that the secrets are very personal to a particular user, and should not be known by others except the server. On the other hand, for example, a wireless local area network (WiFi) needs open encryption algorithms to allow products from different companies to communicate with each other. Otherwise, one company’s WiFi card could not communicate that of another company. However, in our application, the communication is only between a user and a server so that it is good to use secret encryption algorithms, since secret encryption algorithms enhance security by hiding the algorithms/functions. Therefore, we claim that for a very personal communication such as between a user and a server, it is acceptable to use secret encryption algorithms, i.e., algorithms kept as secrets. The function space is infinite with high-order.
Some people may have concerns at this point by claiming that we, trained professionals, cannot provide an easy and secure function, and most users may not either. In fact, this concern is not necessary. Since even a very simple function will be very secure because the attackers do not know what kind of functions the user chose, i.e., functions are kept as secrets instead of keys and the resulting function space is infinite with high-order. Examples of simple functions can be:

- **Flip one bit in the password**;
- **Flip one digit in the password**;
- **Add one to each odd digit and minus one in each even digit**;
- **The first digit of the password is tripled**
- **100x + birthdate, where \( x \) is the real password in an integer form transferred from ASCII codes**;
- **Reversing even bits of the real password in a binary form**;
- **etc.**

User specified functions can be infinite. Since attackers do not know the function forms (i.e., secret encryption algorithms), these simple functions are very secure. Otherwise, it is easy to attack these functions. Note that the user-specified function does not need to be bijective.

We call these simple and secure functions secret little functions. They are useful in our context. One problem is the extra effort in programming the function into the server upon the creation of an account, and human intervention may need.

Also another problem is that secret little functions must use the random number provided by the server, otherwise, it is still subject to Key-logger attacks since the attackers do not need to know the function, but can simply input the same capture inputs again to access.

Advanced users can also define a program to be used.

VI. **Virtual Function with a Helper-Application**

If a helper-application is available for the user, the user needs to type the random salt into the helper-application, and subsequently, the dynamic password is generated by the helper-application. Then the user types in the generated dynamic password in the login screen. In this way, the extra time required is very small and the precision will be one hundred percent correct as long as the user types the correct random salt displayed on the login screen.

This works for the case when the user has a mobile device, such as a cellular phone or PDA. However, such mobile devices are not able themselves to communicate with the server to which the user wants to login. No matter how complex the virtual function is, the helper-application can always generate the correct dynamic password for the user. This case is the most sophisticated one, and it is also the most convenient approach for the user.

For password changing, the user only needs to get a new helper-application after the password change instead of remembering all the changed parts of the virtual password. Note that the server must make the corresponding changes too.

A one-way hash function and many of their functions (such as known encryption algorithms) can serve as virtual functions.

**?? Move this somewhere else??**

If we further assume that the helper-application can communicate with the server, the user only needs to type the random salt in the helper-application, and then the rest of the work is done by the helper-application. The helper-application can generate the dynamic password and submit the login request associated with the user account information, which can be built into the helper-application for the corresponding user. For password changing, if the helper-application can communicate with the server, there is a better way to a change password and make the password more secure, i.e., the helper-application can periodically make the password change request to server and update the corresponding virtual password built into the helper-application. The whole process can be completely transparent to the user.

VII. **Virtual Function without a Helper-Application**

If there is no helper-application for a user, the user needs to calculate the dynamic password from the virtual function with the inputs, random salt and the fixed part of the virtual password. The whole login process may take a little bit longer because it requires the user to perform some calculations. This must work for the user who has no mobile device, so in that case, the virtual function should not be too complicated for human computing.

For password changing, it is similar to traditional password changing. The user can choose a new password, which is the fixed part of the virtual password or a new virtual function, or both. After such changes, the user needs to remember the new virtual password.

The virtual function plays a critical role in the virtual password, especially when the user chooses the option of ‘Use a recommended virtual function” in Fig. 2. There are an infinite number of virtual functions, so that designing an appropriate function is very critical to the success of our scheme.

In order to defend against Phishing, key-loggers, and shoulder-surfing while the system is authenticating the user, this function should meet the following criteria:

1) The function should have some random input provided by the server, which then allows the users to type in different inputs each time they log in the system. This ensures the key logger can not steal the password because the real password is not typed and the typed inputs change each time.
2) The function should be easy for the users. To make the system more secure, we could increase the complexity of the virtual function. However, this resulting function may be very difficult to remember or utilize. The objective is to design less complex but secure virtual functions.

3) The function should be unobservable, i.e., the observed password the user types in for the login session does not disclose hidden secrets; therefore, adversaries cannot use the stolen information to login to the system.

4) The function should be insolvable, i.e., the adversaries should not be able to solve the function with all the potential information they are able to obtain.

5) The above four requirements are used to guide us to design the appropriate virtual functions.

There are many functions which meet all the requirements which we listed above.

VIII. RANDOMIZED LINEAR GENERATION FUNCTION

In this section, we propose one example of a bijective virtual function and analyze how it secures a user password. However, our proposed approach is not limited by these examples. We consider digits here as an example, but our scheme is not limited in the number of digits, nor is it even limited to using only digits.

We propose a function which includes a random number in the function:

\[
B(x) = \begin{cases} 
 k_i = (ax_i + y_i + x_i + c) \mod Z, i = 1 \\
 k_i = (ak, i + y_i + x_i + c + x_{i+1}) \mod Z, i = 2, ..., n
\end{cases}
\]

(1)

where \(a\) is a constant which the user needs to remember but \(c\) is not. The most interesting part of the function is that \(c\) will be a random number which the user randomly picks each time when the user tries to login to the system. Since \(gcd(a, Z) = 1\), the above function is also a bijective function regardless of the \(c\) value. Because \(c\) is also unknown to server, the server knows that \(c \in \{0, 1, ..., Z-1\}\). The authentication could be done as follows.

Let \(B^{-1}(x)\) be the reverse function of \(B(x)\). After the server gets the user’s keyed dynamic password \(k_1, k_2, ..., k_n\) and the fixed part of the virtual password of the user, \(x_1, x_2, ..., x_n\), the server can perform the following verification:

Verify()
\{For each digit \(u \in \{0, 1, ..., Z-1\}\) \{
For each digit in the dynamic password the user typed:
\(w_i = B^{-1}(k_i)\)
if \((w_1w_2...w_n = x_1x_2...x_n)\) return true\}
Return false
\}

The algorithm above guarantees that if the user has input the correct password, the system will grant him/her entrance whatever the random number he/she picked. However, it is also true that for each user, there will be multiple (exactly \(Z\)) acceptable dynamic passwords existing for each specific login session. This may increase the probability that the adversary’s random input happens to be the correct password. However, if the length of your password is long enough, the probability is very small, i.e., \(Z/2^n\), where \(n\) is the length of the password.

A scheme with equation (1) can defend against Phishing, keylogger, shoulder-surfing, and multiple attacks with similar methods discussed later in Section X.B.

[Dr. Li, could please give care explanations about the key space? Here is your note: Since \(y_1\) and \(k\) are known to the adversary after one successful phishing, the attacker can obtain \(ax_1 + x_2 = k\), guess one \((a, x_1)\) to solve \(x_2\). With \(x_2\), we can solve \(x_3\), then \(x_4\), ... and so on. Even if a more careful analysis suggests that \(c\) can't be ignored, then, just guess \(c\) too. The key space will be increased to 400.]

IX. CODEBOOK APPROACH AND REFERENCE FUNCTION APPROACH

Here we propose two approaches for virtual password implementation. They may not be perfect but acceptable in the hostile password phishing environment. For the first approach, some small codebooks will be needed. A codebook should be small enough to be printed on a pocket-sized card or stored in a PDA or a cell phone for the user to carry. It is not impossible but would be unrealistic to ask the user to remember the entire codebook. For the second approach, we design a function that on one hand should be easy to compute with some primitive tool such as paper-and-pencil or a non-scientific calculator. On the other hand, its inverse function should be difficult to compute without knowing some hidden parameters. Apparently, our ultimate goal is to design a zero-knowledge interactive proving protocol but this is impossible given our constraints mentioned earlier. Thus, our next ideal functions would be those that do not give away too much information to significantly compromise the user’s account.

A. Codebook

We first assume that our server has sufficient computing power to run a cryptographically secure RNG (Random Number Generator). This requirement is necessary in protecting the whole system in case some user loses their codebook to a wrong hand, so that the system will not be compromised and the user can easily ask for a new codebook without changing the parameters of the RNG. Note that, LCG (Linear Congruential Generators) is not such cryptographically secure RNG.

Our first codebook is rather straightforward. In the setup session, the user decides the length of the password, \(n\). Then, the server gives \(n\) 10-digit random numbers. Suppose we are doing this for protecting 4-digit PIN, i.e. \(n = 4\). The server outputs four random numbers \(X_0, X_1, X_2, X_3\), and \(X_4\) each has 10 digits. Let \(x_{0,0}, x_{1,0}, x_{1,1}, x_{1,2}, ..., x_{1,9}\) denote the ten digits of \(X_1\). The user’s codebook is given as follows:

\[
X_{0,0}, X_{0,1}, X_{0,2}, ..., X_{0,9}
X_{1,0}, X_{1,1}, X_{1,2}, ..., X_{1,9}
...\]


three successful attacks, the adversary’s odd of getting into the system increases to 1.87×10^{-2}. In this case the victim’s account is still relatively safe if we require the server to lock the account after attempting to login with invalid password a few times. We may also increase the length of the password to resist more attacks. For example, if we use ten digits, the chance of compromising the account after three successful phishing attacks is about the same as a 4-digit PIN code under a phishing free environment. However, we should take every precaution and assume that five or more successful phishing attacks can happen to a careless user. To have a password with 20 digits is not realistic. Instead, we can increase the symbol size by allowing letters and some special symbols to be used in the passwords. In practice 64 is a reasonable symbol size. We have the following results.

<table>
<thead>
<tr>
<th>symbol size s = 64</th>
<th>n: length of the password</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.96E-08</td>
<td>1.46E-11</td>
<td>3.55E-15</td>
<td>8.67E-19</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9.84E-07</td>
<td>9.76E-10</td>
<td>9.68E-13</td>
<td>9.60E-16</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5.03E-06</td>
<td>1.38E-09</td>
<td>2.53E-11</td>
<td>5.68E-14</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.60E-05</td>
<td>6.39E-08</td>
<td>2.56E-10</td>
<td>1.02E-12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.92E-05</td>
<td>2.45E-07</td>
<td>1.53E-09</td>
<td>9.59E-12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8.14E-05</td>
<td>7.34E-07</td>
<td>6.62E-09</td>
<td>5.97E-11</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.51E-04</td>
<td>1.85E-06</td>
<td>2.28E-08</td>
<td>2.80E-10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2.58E-04</td>
<td>4.14E-06</td>
<td>6.64E-08</td>
<td>1.07E-09</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4.13E-04</td>
<td>8.40E-06</td>
<td>1.71E-07</td>
<td>3.47E-09</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6.30E-04</td>
<td>1.58E-05</td>
<td>3.97E-07</td>
<td>9.97E-09</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9.23E-04</td>
<td>2.81E-05</td>
<td>8.53E-07</td>
<td>2.59E-08</td>
<td></td>
</tr>
</tbody>
</table>

According to the table above, if symbol size is increased to 64, the security level of 4 digit passwords after 5 successful phishing attacks is still at the level of conventional 4-digit PIN code under a phishing free environment. In practice, it is not likely a user will satisfy the phisher more than 5 times without getting suspicious. Note that our concern is very different from the chosen (or known)-plain text attack in the context of cryptography; a large amount of plain-cipher text is not available to the phisher.

B. Reference Function Approach

Let Σ denote the alphanumeric set, where alphabets are case insensitive. Each element in Σ is coded by 0=0, 1=1 ... 9=9, A=10, B=11 ... and Z=35 for arithmetic. Here we present a reference function (RF): Σ→Σ using the fixed part of the virtual password as references. RF does not have to be bijective as we believe that this restriction will benefit the adversary in narrowing down the possibilities. Since the server and the user share the same hidden parameters of the function, the two parties can obtain the same result while the adversary has little chance to guess the password. Let \( x=x_1x_2...x_n \) be the fixed and hidden part of the virtual password, provided that we require \( n \) to be a prime number no less than 7. This length restriction is required for security reasons that we will explain in a moment. When the user tries to login to the system, the
system presents \( n \) random alphanumeric digits \( r = r_1 r_2 ... r_n \). Let \( k = k_1 k_2 ... k_n \) be the dynamic password that the user needs to input. \( RF(x, r) = k \) is computed as follows:

\[
\text{for each } i = 1 \ldots n, \quad k_i = r_i x_{(i/\text{mod} n)+1} \mod 36 \quad (2)
\]

Consider the following example. Let \( n = 7 \), \( x = ABCD123 \), and let \( S \) denote the set of alphanumeric in \( x \), i.e., \( S = \{A, B, C, D, 1, 2, 3\} \). Suppose that the server presents \( r = 1234567 \) for the user to compute the virtual password. We have the following calculation:

<table>
<thead>
<tr>
<th>( X_i )</th>
<th>code</th>
<th>( r_i )</th>
<th>( x_{r_i} )</th>
<th>( X_{a_i} )</th>
<th>( k_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>A</td>
<td>10</td>
<td>1</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>B</td>
<td>11</td>
<td>1</td>
<td>B</td>
<td>M</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>C</td>
<td>12</td>
<td>3</td>
<td>B</td>
<td>X</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>D</td>
<td>13</td>
<td>4</td>
<td>D</td>
<td>G</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>( x_6 )</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>( x_7 )</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>A</td>
</tr>
</tbody>
</table>

The 5th column of the table \( a_i = (s_{r_i} \mod 7) + 1 \) is computed for the reference of \( x \) that will actually be used in computing \( k \). The user should input DMXGACY as the password and this is what the server is expecting for authentication.

What happens if the user is linked to a Phisher’s page? A successful Phishing as well as the other two methods, shoulder surfing and key-logger, will make \( r \) and \( k \) available to the adversary. Thus, the adversary can recover the set of elements in the 6th column of the above table by \( x_{a_i} = r_i^{-1} k_i \mod 36 \). Let such set be \( S' \), and thus \( S' \subseteq S \). However, the positions of these digits will not be revealed no matter how many rounds of successful phishing have been conducted by the adversary, because all he/she can get is a subset of \( S \). For example, with \( r = 1111111 \), the adversary will have \( S' = \{B, C, D, 1, 2, 3\} \). In the worst case, the adversary may get \( S' = S \), but the actual \( x \) is still protected to a certain amount by the number of permutations of \( S \). To make this amount more significant, we suggest considering \( x \) to have at least 11 digits. In this case, even when \( S \) is recovered, we still have more than 35 millions permutations for the phisher to guess. We suggest the length of \( x \) being a prime number in order to be sure that every digit in \( x \) has a chance to be used. For example, if the length of \( x \) is even and \( r \) contains only even digits, then only the digits at even positions in \( x \) will be used for \( k \).

RF is much stronger than the codebook approach since the function using a codebook cannot survive 10 successful phishing attacks while the RF can survive arbitrary number of successful phishing attacks.

X. LESSONS LEARNED IN DESIGNING VIRTUAL FUNCTIONS

Before we finally identified the two approaches proposed in the previous two sections, we in fact considered many candidates of virtual functions. In this section, we introduce some lessons learned and mistakes which we made at the early stage of designing virtual functions. We share our experiences, since we believe that these experiences are useful. We start with very simple cases followed by more complex ones.

A. Lesson One

At the beginning we pursued bijective functions as virtual functions since we thought that since bijective functions had reverse functions, it was easy to verify a user by the server. Later, we believed that it was not necessary for virtual functions to be bijective.

We began by using simple operators, such as plus or minus some secret value, e.g., \( B(x) = (x+y) \mod Z \), which is easy to use, where \( x \) is the fixed part of user’s virtual password and \( y \) is the secret value. We immediately excluded this case since it is easy to attack [26] and violates the first requirement from the previous section because it has no random variable. We concluded that we could use a simple linear function that include a random variable, such as \( B(x) = (a*x+y) \mod Z \), where \( x \) is the fixed part of the user’s virtual password, \( y \) is a random number the system provides to the user in each login session and \( a \) is kept as secret. This function involves the random factor, but it violates the unobservable rule. This is because once the adversary obtains the function output \( k \) and random factor \( y \) (through shoulder-surfing or Phishing), they can easily login to the system based on the stolen information. This is because with \( k \) and \( y \), they can deduce \( (a*x) \mod Z = (k-y) \mod Z \). Then the adversary can login to the system with \( (a*x+y') \mod Z \), where \( y' \) is a new random number.

For example, suppose that you have a virtual password 123 and the function \( (3*x+y) \mod 10 \). In the login session, the system provides the random number, 456. You calculate \( (3*4+6) \mod 10 = 7 \), \( (3*2+5) \mod 10 = 1 \), and \( (3*3+6) \mod 10 = 5 \), so that it is 715. Once the adversary steals the random number and the dynamic password the user input, the adversary can determine \( 3*x_1 = 3 \), since \( (7-4) \mod 10 = 3 \), \( 3*x_2 = 6 \) since \( (1-5) \mod 10 = 6 \), and \( 3*x_3 = 9 \) since \( (5-6) \mod 10 = 9 \). Now, the adversary is able to access the system. If the system generated random number is 789, the adversary can enter the system by providing the password 048, where \( 0 = (7 + (7-4) \mod 10, 4 = (1 + (8-5)) \mod 10, \) and \( 8 = (5 + (9-6)) \mod 10 \). This is the correct the real password.

Then, we considered the function \( B(x_i) = ax_i + y_i \mod Z \), where \( x_i \) is one digit of the password, \( y_i \) is a random number provided by the server, and \( a \) is kept as secret. However, we discovered that this function is subject to shoulder-surfing attacks. For example, if a hidden camera records a user’s activity (actions of key boards), then it is easy to obtain \( ax \), if \( y \) is known.

?? How is this different from previous functions??

B. Lesson Two

Next, we included added a constant factor to the linear function:

\[
B(x_i) = \left[ a(x_i + y_i) + c \right] \mod Z \quad (3)
\]
where \( a \) and \( Z \) are relatively prime, \( x_i \) is one digit from the fixed part of the user’s virtual password, \( y_i \) is one random digit provided by the system, and \( a \) and \( c \) are the constant factors of the linear function, which the user has to remember. The \( B(x) \) is a bijective function if and only if \( \gcd(a, Z) = 1 \) \([28]\), where \( \gcd \) denotes the Greatest Common Divisor function.

In fact, we can prove that this function can stand for Phishing, key logger, and shoulder-surfing attacks by the follow security analysis. Let \( x_1,x_2,...,x_n \) and \( [a(x_i+y_i)+c] \mod Z \) denote the user’s virtual password, where \( y = y_1y_2...y_n \) is random number provided by the system/server.

• **Defense against Phishing:** For Phishing, once a user is lured to type in the dynamic password \( (x_1,x_2,...,x_n) \) in the faked page, then the dynamic password will be recorded by the adversaries. However, we claim that this does not help the adversaries to figure out the virtual password of the user, because it is impossible to get the solution of \( x_1,x_2,...,x_n \) a, and c based on the information they have stolen. We can present the digits the Phisher caught in the form of equations and there are \( n \) equations the Phisher can build as: \([a(x_i+y_i)+c] \mod Z=k_i\), \( (for i=1,...,n)\), where \( y_1 \) to \( y_n \) and \( k_1 \) to \( k_n \) are known by the Phisher. However, since there are \( n+2 \) variables \( (a, c \) and \( x_i \) to \( x_n \) ), but only \( n \) equations, it is impossible to solve the equations to get the solution. Therefore, we claim that our scheme is phising-proof.

• **Defense against the key logger:** The key logger code logs all the key strokes at the operating system level so that such logs are delivered to some adversaries who analyze what the victim has keyed in their system, and then try to extract the user password. Such a key logger will be very effective if the user typed their password in an unsafe machine on which the key logger is installed. In our scheme, the key logger can still catch the entire user’s key strokes, but they still need to solve the above mentioned \( n \) equations, where the only variables the logger can be aware of are \( k_i \) to \( k_n \). We also assume that the user does not add some noise into the logger as \([17]\). If noises are created, we claim that the adversaries are not able to have the equations at all. Even if they build the equations; they cannot solve the function because the available knowledge for the adversaries is not enough.

• **Defense against the shoulder-surfing:** To protect user from the shoulder-surfing, we assume that the watchers have a good memory or they use some other devices, such as a camera, to record all the information that the user will use, including the random digits that the system provides in the screen and the keys the user types into the password field. With the help of the virtual password, at the prospective of the watchers, the information available to them are the dynamic password \( k_1k_2...k_n \) and the random digits \( y_1y_2...y_n \). The watcher will face the same challenge that the Phisher encounters, and will have difficulty in solving the above mentioned \( n \) equations.

We have claimed that function (3) will protect the user from a Phishing attack, key loggers, and shoulder-surfing. However, we later found out that function (3) has at least one drawback, e.g., it cannot stand for multiple attacks as follows.

• **Challenge for multiple attacks:** if the user is lured to try to login to any same phishing website more than twice, it will leak his/her password. Suppose that Bob has a set up a Phishing website abc.com and Alice is a user of the website with password \( k_1k_2...k_n \) with equation (3). Now if Alice tries to login twice to the Phishing website abc.com, Bob can compare the dynamic passwords which Alice input, and then Bob can easily figure out how to login to the real website with Alice’s account. The reasons are explained as follows. For any given \( i \)-th digit of the fixed part of Alice’s virtual password, if Alice has tried more than twice to login to the fake website, then Bob could obtain the two equations below: \([a(x_i+y_i)+c] \mod Z=k_i \), \( (for \ (i=1,...,n) \) \). Now Bob can know that \( [a(y_i-y_i')]+c \mod Z=k_{i}' \), \( \) where \( k_{i}' \) is the first time Alice typed in the \( i \)-th digit in the fake website, \( y_i \) is the \( i \)-th random digit provided by the fake website, and \( y_i' \) is the \( i \)-th digit the system will display on the screen, which Bob needs to login to Alice account.

Such leaking can also occur in shoulder-surfing if the watcher can record all the information for the same victim twice. However, if the attackers do not know the functions used, i.e., equation (3), then the attackers cannot figure out the real password. Furthermore, during the registration phase, if we adopt a user-defined function or a user-defined program shown in Fig.2, these attacks can be totally avoided, as explained in Section V.

We continued in our attempts to design a virtual function instead of considering only user-defined functions. The result was to make equation (3) more complex. We revised equation (3) so that is uses the value of a digit in the dynamic password to calculate a subsequent digit in the dynamic password. We called this new function a **randomized linear generation** function as follows.

\[
B(x_1) = \begin{cases} 
K_1 = (a.x_1 + y_1 + c) \mod Z \\
K_i = (a.K_{i-1} + y_i + c.x_i) \mod Z 
\end{cases}
\]

where \( a \) is a constant which the user needs to remember but \( c \) is not. The most interesting part of the function is that \( c \) will be a random number which the user randomly chooses each time the user tries to login to the system. Since \( \gcd(a, Z) = 1 \), the above function is also a bijective function regardless of the \( c \) value. How the server authenticates the user’s validity is different from the previous scheme based on equation (3). Because \( c \) is also unknown to server, the server knows that \( c \in \{0,1,...,Z-1\} \). The authentication is the same as that for equation (1) and can be done as follows.
Let $B_k^{-1}(x)$ be the reverse function of $B(x)$. After the server gets the user’s keyed dynamic password $k_1k_2...k_n$ and the fixed part of the virtual password of the user, $x_1x_2...x_m$, the server can perform the following verification:

Verify()
\{ For each digit $u \in \{0,1,...,Z-1\}$ \{
  For each digit in the dynamic password the user typed
  \{ $w_i = B_k^{-1}(k_i,u)$
  \} if $(w_1w_2...w_n = x_1x_2...x_m)$ return true
\}
Return false
\}

The algorithm above guarantees that if the user has input the correct password, the system will grant him/her entrance whatever the random number he/she picked. However, it is also true that for each user, there will be multiple (exactly $Z$) acceptable dynamic passwords existing for each specific login session. This may increase the probability that the adversary’s random input happens to be the correct password. However, if length of the password is long enough, the probability is very small, i.e., $Z/2^n$, where $n$ is the length of the password.

The scheme with equation (4) can defend against Phishing, keylogger, shoulder-surfing, and multiple attacks as follows.

- **Defense against Phishing, keylogger, and shoulder-surfing**: It protects the user’s from password stealing based on the same theory that the adversary cannot solve the function because the adversary does not have enough information. We only use Phishing as an example here. We now list the following equations: $K_1=(ax_1+y_1+c) \mod Z$ and $K_i=(ak_{i-1}+y_i+c) \mod Z$ ($i=2,...,n$). For the Phisher, the $c$, $a$, $x_1$, ..., $x_n$ are unknown, and they only know the $k_1k_2...k_n$ and $y_1y_2...y_m$ so that they cannot solve the function to get the solution. These similar unsolvable equations exist against key loggers and shoulder-surfing.

- **Defense against multiple attacks**: We claim that multiple dynamic password leaking in the virtual password scheme with a linear generation function can be secured using a random number. If an adversary has tricked a user into logging into his fake website twice, the adversary obtains $k_1=(ax_1+y_1+c) \mod Z$ and $k_i=(ax_i+y_i+c') \mod Z$, where $a$, $m_i$, $c$, and $c'$ are unknown to the adversary, and then what information the adversary can figure out is the $(c'c) \mod Z=(y_1c'-y_1c)k_i$. Since the $c$ and $c'$ were randomly chosen by the user, $(c'c)$ does not provide any information. If the adversary cannot work out some clue about the first digit of the dynamic password $k_1$, he/she cannot find about $k_2$ and later digits in the dynamic password. Therefore, using the linear function with a random number can remove the possibility of multiple dynamic password attacks.

However, at this moment, we discovered that both the scheme with equation (3) and the scheme with equation (4) suffer a drawback: a small key space.

For equation (3), i.e., $B(x) = (ax+y+c) \mod Z$, if $Z=10$, the choice of $a$ is 4 since $\gcd(a,Z)=1$ and the choice of $c$ is 10. Therefore, the total number of choices for pair $(a, c)$ is 40, i.e., the key space is 40. With each choice of pairs $(a, c)$, and $y$, an attacker can guess 40 choices of $x$ values.

The key space of size 40 is not acceptable. It is different from 40 phishing attacks, since a human is not likely to tolerate more than 2 or 3 phishing attacks without being able to get into the system. But the 40 keys is for the adversary to login the system, not to phish a human. On average, 20 tries can let him get into the system. Even if the system is designed to lock the account after 3 wrong passwords are entered, the adversary can collect 10 successful phishing results from 10 different users. This can be easily done. Then, for each account, he can try 3 passwords, and the chance he could get onto at least one account is higher than $\frac{1}{2}$, (precisely, $1-(37/40)^3$).

Equation (4) also suffers the same problem of a small key space.

Hence, we considered a little more complex function as follows.

\[ B(x_i) = \left[ a(x_i+y_i) + x_{[i+1] \mod m_i} + 1 \right] \mod Z \] (5),

where $n$ is the length of the fixed part of the virtual password.

We can easily demonstrate that the above function protects the system from Phishing, key logger, and shoulder-surfing attacks with similar methods from those for equation (3).

However, the key space using equation (5) is still 40 after one successful phishing. The adversary can guess $a$ and $x_i$, and then he can solve $x_{i+1}$, then $x_{i+2}$, and so on. In other words, each pair of $(a, x)$ determines the whole $x$.

Furthermore, similar to the scheme in equation (3), we can prove that the above function is subject to multiple attacks. Such leaking can also occur in shoulder-surfing if the watcher can record all the information for the same victim twice.

We then considered the following function

\[ B(x_i) = \begin{cases} K_1 = (ax_1+y_1+x_2+c) \mod Z, i = 1 \\ K_i = (ak_{i-1} + y_i + x_i + cx_{i+1}) \mod Z, i = 2,...,n \end{cases} \] (6),

where $c$ is a user-chosen random number.

The random constant $c$ is irrelevant when $i>1$ since the adversary can pick up any one just like the user can. So, the adversary can choose $\theta$ to cancel the effect.

Finally, instead of $cx_{i+1}$, we chose $c+x_{i+1}$ to result in equation (1).

C. Lesson Three

Before we come up with the reference function scheme proposed in last section, i.e., equation (1) and its related
method, we considered several other functions introduced in this subsection.

For the first attempt, we required \( n \) to be at least 8. The minimal length 8 is required for security reason as we will explain in a moment.

For each \( i = 1 \ldots n \), \( k_i = (x_i \mod n + r) \mod 36 \) \( (7) \)

Consider the following example, let \( n = 5 \), which is not secure in practice, but we make it small for demonstration purposes. Let \( X = \text{A9C48} \) and the system present \( R = 12345 \) to the user for computing the virtual password. We have the following calculation:

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>code</th>
<th>code mod 5</th>
<th>( X_{x_i} )</th>
<th>( r_i )</th>
<th>( k_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>0</td>
<td>A</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td>8</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>2</td>
<td>C</td>
<td>3</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>

A successful phishing as well as the other two attack methods will make \( r \) and \( k \) available to the adversary. Thus, the adversary will recover the set of elements in the 4th column above. Let such set be \( S \), i.e., \( S = \{A, C, 4, B\} \). We have \( S \subseteq \{A, 9, C, 4, 8\} \), the set of alphanumeric digits of the hidden parameters. However, the positions of these digits will not be revealed no matter how many rounds of successful phishing have been conducted by the adversary, because all he/she can get is the same subset \( S \). In other words, the hidden part of the virtual password is still protected by its possible permutations. In the case of \( n = 5 \), the key space after some successful phishing is at least of size 120. This is not acceptable, but it will increase exponentially in accordance with the key length. We suggest \( n \geq 8 \) to have a key space of at least 40K in order to survive after some successful phishing; when \( n = 11 \), we have more than 40 million.

The drawback of the above scheme is explained as follows. Note that, the random digit \( r \) does not play an essential part in our scheme, but we still suggest it here to hide \( S \) from impulsive shoulder-surfing. Also note that, having \( r \) be a part of reference calculation is not a good idea because it will increase the chance for the adversary to locate the position of \( S \) by using some differential technique. For example, the adversary can compare the difference between the two results of using \( r = 00000 \) and \( r = 10000 \), respectively, to guess where \( x_i \) points.

Furthermore, there is an obvious loophole in the above scheme. The adversary need not know the actual position of \( x_i \). The arrangement in the 4th column is enough to input correct code for any other random number \( r \) afterwards. Therefore, the above scheme cannot survive even one successful phishing attack and it seems we are back to our original scheme where we need two sets of hidden passwords. This apparently is not a valid idea. [Dr. Li could you please check the above statement?]

The second attempt is to use the following function.

For each \( i = 1 \ldots n \), \( k_i = (r_i + x_i \mod n + r_i) \mod 36 \) \( (8) \)

It is easier to compute this function, but it cannot resist \( n \) successful phishing attacks, where \( n \) is the length of the password. The phisher can try \( r_1 = 1, 2 \ldots n \) to find out \( k_i k_2 \ldots k_{n} \). Furthermore, \( x \) is one of the \( n \) permutations of \( k_i k_2 \ldots k_{n} \). If we allow the system to choose 0 in the random digits, \( x_i \) will be revealed. But if the length of \( x \) is big enough, this should not be a concern; or we can simply restrict 0 from the random digits.

[Dr. Li could you please give more details for this part?]

With all the experience we had, we finally came up with the proposed scheme in equation (2).

\[ \mu \text{TESLA} \] is an authentication scheme, which was originally designed for sensors to authenticate a broadcast message sender in a sensor network based on a public one-way hash function \( F \).

\[ \mu \text{TESLA} \] [34] is an authentication scheme, which was originally designed for sensors to authenticate a broadcast message sender in a sensor network based on a public one-way hash function \( F \).

We could use the methodology to defend against Phishing attacks or to authenticate the server before re-keying. We adjust it with the assumption that the server side and client side will choose the same public hash function; still we discuss how it works in the registration phase, sign-on phase, and password change phase.

In the registration phase, upon a registration request, in addition to preparing the general password, user id, and other information, the server needs to generate a chain key \( K_m, \ldots, K_0 \) by randomly choosing the \( K_m \) and then producing the \( K_{m,0} = F(K_m) \) where \( n = 0, 1, \ldots, m \). The server will then pass the \( K_0 \) to the client.

In the sign-on phase, once the sign-on request arrives at the server side, the server should present the sign on screen to the clients and it also needs to provide the authentication code, which will be encrypted by the latest key. For example, the \( n \)-th time the user signs in the system, the server produces the authentication code as \( E_{K_{m,0}}(K_0) \) and passes this to the client. Meanwhile, when the client receives the package, it needs first to decrypt the authentication code with the current key (i.e., \( K_{m,0} \)) which it holds to get the \( K_m \) and then using this \( K_m \) to verify if this sign-on screen is from the right server in the following way: if \( F(K_m) = \text{Key} \) it currently holds, it is
verified and the currently held key is updated to be $K_n$; otherwise, it is denied.

In this way, the client can verify the server and protect the client from Phishing attacks because the Phisher has no knowledge about the $K_0, \ldots, K_m$, so that it isn’t able to fake the authentication code. Furthermore, the server should use the latest used key to encrypt the current key, since if the authentication code is not encrypted, the Phisher could pretend that they are clients, and try to log in the system. Then the server presents the login screen along with the new authentication code $K_n$ which makes it difficult for the Phisher to fake a login screen with the correct authentication code to lure the client.

Because the number of keys from $K_0$ to $K_m$ is finite, the server will use up all the authentication codes some day, although we can choose a very large value of $m$. The server and client should build a scheme to regenerate their authentication code, which we refer to here as re-keying of the authentication key refreshes. This is an easy job and can be conducted once both the client and the server verify each other. The server needs to generate a new chain of keys as when it did in the user registration phase, and to deliver the first of the keys to the client.

This μTESLA scheme will work effectively to shield the clients from Phishing attacks and it could be used to protect the user’s password together with our virtual password scheme.

XII. IMPLEMENTATION AND EVALUATION

In order to implement the virtual password scheme to safeguard users when they are surfing online, we implemented the scheme, and demonstrate that a little human computing can defeat Phishing, key logger and shouldersurfing attacks. In this section we will evaluate our practical implement of the virtual password scheme.

A. Screenshots of System Implementation

We just implement a simple scheme for illustration purposes, e.g., the randomized linear function. With the consideration of user usability, we set $Z=10$, so that the available values for $a$ are $\{1, 3, 7, 9\}$. This does not decrease the scheme’s security due to the limited value of $a$, since in the linear generation function, the value of the dynamic password will rely heavily on the random $c$. The function $K_n = (a.K_{n-1} + y_a + X_n + c \cdot X_{n-1}) \mod 10$ may be too difficult for users to calculate in their mind, especially since $K_{n+1}$ is hard to remember. In our implementation, we will allow the system to display the password file without marking the content as “*”, which will make it easier for the user to know the previous digit he/she has entered. In Fig. 3, we demonstrate a testing website in which a virtual password scheme was implemented.

Even though such a calculation is a little complicated for some people, our helper-applications could relieve the users of this required human computing. In Fig. 4 and Fig. 5, we implement two versions of such helper-applications for a personal computer and a mobile device, respectively.

B. Password-security survey for users’ responses

Currently, most of the websites allow a user to have only one fixed unique password; however, in our scheme, the password is dynamic and a user needs to make some computations for each login if the user has not installed the helper-application, which is significantly different from the traditional way that the user just inputs a password. The traditional one may seem more comfortable for the user, but the price of such comfortableness is that the password could
be stolen by adversaries. If considering the fact that users tend to pick passwords that are usually used in cross-systems for easy recall, or those related to the users’ privacy, such as DOB, nick name, and so on, the traditional password is more vulnerable. Although it is tedious for users to make some calculations each time to login to the system, the well trained user can finish the entire login process in a short time.

We distributed a survey (Figs.6 -12) to collect users’ responses for our system implementation with a total of 86 responses. We found that the respondents have an average of 10 or more online accounts, as shown in Fig.6, but the majority of them are unaware of how to defend against Phishing, Key loggers, and Shoulder-surfing. Meanwhile, many of the respondents have no idea about what the three attacks are as illustrated in Figs 7-9. This severe fact indicates that it is urgent to take some action to protect innocent users from those types of attacks. As shown in Fig.10, we also found that most of the people could complete the single digit calculation easily, without help from the calculator. This makes our virtual password scheme applicable even for people who do not have any mobile devices with them. As we described in the previous section, we could design some simple bijective function as a randomize generation function to allow for easy human computing. In Figs. 11 and 12, respondents express their demand for better secure internet and most of them would accept the cost of spending a little bit extra time to sign onto the system with the improvement in password security. We argue that the extra time will be acceptable for most of the people based Fig. 10, and furthermore, if the helper-application is available for users, the extra time will be very small, and there is no extra time at all if the mobile device can communicate with the server.

![Fig. 6 How many online account you have?](image1)

![Fig. 7 Knowledge about Phishing attack](image2)

![Fig. 8 Knowledge about Key logger attack](image3)

![Fig. 9 Knowledge about Shoulder-surfing Attack](image4)

![Fig. 10 How comfortable to do the single digit calculation?](image5)

![Fig. 11. Current internet secure enough to protect your password?](image6)

![Fig. 12 Would you like to improve your password Security with a little bit extra time?](image7)

C. Usability Test

We test whether the approach is easily used by users or not. In order to test the usability of our scheme, we
conducted a usability test with six student volunteers. In our testing, each volunteer was asked to try to login to our test website for two rounds. For the first round, they needed to calculate the password by themselves, and for the second round, they used a helper application to calculate the password for them. The complete each round ten times and recorded the time it took them to complete their login. Figs. 13 and 14 demonstrate the usability testing results:

![Graph showing Average Time to Log In](image)

**Fig. 13 Average log in time**

![Graph showing Successful Rate of Log In](image)

**Fig. 14 Average Successful Rate (in percentage)**

From Figs. 13 and 14, we can see that the user time to login to the system if they do not utilize a helper application can vary depending on their ability to perform simple calculations. The users will take an average of 47 seconds to login to the system, but with help from the helper application which runs at the mobile device or lap top, users will only take around 15 seconds to login to the system. In another aspect, without a helper application, the login success rate is around 75%, but with a helper application, the login success rate will be always 100%. We argue that it is worth it to take a little bit longer (an extra 10 seconds with a helper application) to login to the system, if such a login will be guaranteed secure. This is especially important when a user logs into some important system via the internet, such as an online banking account or credit card account.

XIII. CONCLUSION

We discussed the challenges of protecting users’ passwords on the internet and presented some related work in this field. We discussed how to prevent users’ passwords from being stolen by adversaries. We proposed a virtual password concept involving a small amount of human computing to secure users’ passwords in on-line environments. We proposed differentiated security mechanisms in which a user has the freedom to choose a virtual password scheme ranging from weak security to strong security. The function/program is used to implement the virtual password concept with a trade off between security and complexity requiring small amount of human computing. However, since simplicity and security conflict with each other, it is a challenging task to achieve both if possible. We further proposed several functions serving as system recommended functions and provide security analysis. We analyzed how the proposed schemes defend against phishing, key logger, shoulder-surfing attacks, and multiple attacks. In user-specified functions, we adopted secret little functions, in which security is enhanced by hiding secret functions/algorithms. In conclusion, user-defined functions (secret little functions) are a better way.

In the future, we plan to study how to design smarter functions to alleviate the computation-burden of the user. We would also like to develop some small applications with virtual passwords built-in, which will be able to run at a customer’s wireless device, such as a cellular phone or a PDA. Using such an application, the user just needs to input the system random digits which the system provides, and then the dynamic password is automatically calculated for the user.

REFERENCE

1) To call the function “virtual function” may disturb OOP (Object-Oriented) people. ==> "virtual password function" if we want, make an acronym — VPF.

2) Also, unless we give a special name to the function, let's use F to denote the function. (we no longer want it to be bijective, right?)

3) If there are secret parameters such as a and c in, for example, 
   \[ S_a \ast \ x_i \ast c_i \mod Z, \] let's call them "the hidden parameters of VFP".
   Also, make them explicit in the name of the VFP, for example, $F_{\{a,c\}}$.

4) Using upper case X in italic for vector, little case with subscription in italic for digits. That is, $x = x_1 \cdot x_2$. Also, function mod should not be in italic.

5) X (used in present papers) is called "the hidden password".

6) Let the result of the VPF be called "virtual password" (because they are not real). Let's use V in italic to denote such thing.

7) Y (using in the present papers, except in the RF function section), ==> R; This reflects that it is a random number

8) Thus, we can abuse the notation by alternatively use/define $F(X,R) = V$ or $F(x_{1:1}, r_{1:1}) = v_{1:1}$, without further describing what they mean, since it should be clear to the reader. (We reserve the subscription of F for its hidden parameters, if any).