

Secret Little Functions and Codebook for Protecting Users from Password Theft

Yang Xiao⁺, Chung-Chih Li^{*}, Ming Lei⁺, and Susan V. Vrbsky⁺

⁺Department of Computer Science, The University of Alabama,

^{*}School of Information Technology, Illinois State University

{mlei, yangxiao, vrbsky}@cs.ua.edu, cli2@ilstu.edu,

Abstract—In this paper, we discuss how to prevent users' passwords from being stolen by adversaries. 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 codebook approach to serve as system recommended functions and provide a security analysis. For user-specified functions, we adopt secret little functions, in which security is enhanced by hiding secret functions/algorithms.

I. INTRODUCTION

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.

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. *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. *Password Stealing Trojan* is a program that contains or installs malicious code. There are many such Trojan codes that have been found online today. Key loggers capture keystrokes and store them somewhere in the machine, or send them back to the adversary. *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.

In this paper, we discuss how to prevent users' passwords from being stolen by adversaries. 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 codebook approach to serve as system recommended functions and provide a security analysis. For user-specified functions, we adopt secret little functions, in which security is enhanced by hiding secret functions/algorithms. We further propose a scheme to adopt μ TESLA to be used for re-keying and to defend against Phishing. 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.

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.

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.

Any of the schemes above cannot prevent against Phishing, Trojan horse, and shoulder-surfing at the same time.

III. PROVIDING DIFFERENTIATED SECURITY THROUGH A VIRTUAL FUNCTION

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 ψ to ψ , where the ψ is the letter space which can be used as passwords. We have $P=(F, B)$ and $B(F, R) = P_d$, 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_d 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_d), where ID is user ID. On the server side, the server can also calculate P_d 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's ID, and compute P_d , 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.

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.1. No matter whether a virtual function is used or not, the user is required to input the read password and ID in Step 2 of the registration.

In Fig. 1, 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.

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.

IV. SECRET LITTLE FUNCTION (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.

Please choose your PIN registration approach among the followings

- Default: do not use a virtual function;
- Use a recommended virtual function
 - Use function $B=XXX$.
 - Use function $B=YYY$.
 - Use function $B=ZZZ$.
- Use a user defined function (Note that user and server share a common function specified by the user): _____
- Indirect-specified system function, please choose a security degree: Low , Medium , High , or Very High
- Use a user defined program (in C or Java) (Note that user and server share a common program specified by the user): _____

Fig.1 A screenshot of user registration: Step 1

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.

V. CODEBOOK APPROACH

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. 1. Although there are literally infinite many functions to use, to choose a suitable one by no means should be arbitrary without considering hostile environment. In other words, 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:

$$x_{(i,a)} x_{(i,b)} x_{(i,c)} x_{(i,d)}$$

- 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.

The four requirements above should be used to guide us in designing appropriate virtual functions. While there are many zero-knowledge protocols available for secure authentication, they are not applicable to our applications because they all require significant computing power from the users. On the other hand, many simple functions that intuitively seem to be good candidates for our purpose, but after more careful analysis we can show they are problematic.

Here we propose one approach for virtual password implementation. It may not be perfect but acceptable in the hostile password phishing environment. In the proposed 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. Apparently, our ultimate goal is to design a zero-knowledge interactive proving protocol but this is impossible given our constraints mentioned earlier.

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 , and X_3 , each has 10 digits. Let $x_{(i,0)}, x_{(i,1)}, x_{(i,2)}, \dots, x_{(i,9)}$, denote the ten digits of X_i . The user's codebook is given as follows:

$$x_{(0,0)}, x_{(0,1)}, x_{(0,2)}, \dots, x_{(0,9)}$$

$$x_{(1,0)}, x_{(1,1)}, x_{(1,2)}, \dots, x_{(1,9)}$$

$$x_{(2,0)}, x_{(2,1)}, x_{(2,2)}, \dots, x_{(2,9)}$$

$$x_{(3,0)}, x_{(3,1)}, x_{(3,2)}, \dots, x_{(3,9)}$$

It is up to the user to decide how to store or memorize the codebook. To login the system, the system will present a 4-digit random number $R = abcd$, where each letter represent a digit. The virtual password for the user to key in would be:

For security analysis, we consider the phishing attack only because it is the most aggressive attack where the adversary can control the random number R . For each attack, the phisher will provide a fake random number R to the victim. If he succeeds, the adversary will get four corresponding digits in the code book. As the result, the chance the adversary can correctly guess one digit of the password is the chance the system asks for the same position plus the chance the system asks for the other nine positions and the adversary guess it rightly. That is,

$$\frac{1}{10} + \frac{9}{10} \times \frac{1}{10} \approx \frac{1}{5}$$

In other words, the chance for the adversary to break into the victim's account after one successful phishing is $(0.2)^4 = 1/625$. It is likely that the adversary will conduct more than one attack and the victim may not be aware of the situation in first few rounds of phishing. To maximize the information to gain, the adversary will ask different positions in each phishing. Let p be the number of successful phishing attacked at the same user. Also, let n be the length of the password and s the number of different symbols for a digit (in the present example, $s = 10$). We have the following formula for the chance the adversary may get into the victim's account.

$$\left(\frac{p}{s} + \frac{s-p}{s} \times \frac{1}{s} \right)^p = \left(\frac{1+p}{s} - \frac{p}{s^2} \right)^p$$

We obtain the following result of using this codebook.

TABLE 1: THE CHANCE OF BREAKING THE PASSWORDS UNDER PHISHING ATTACKS (SYMBOL SIZE = 10)

symbol size $s = 10$		n: length of the password			
		4	6	8	10
p: number of successful phishing attacks	0	1.00E-04	1.00E-06	1.00E-08	1.00E-10
	1	1.30E-03	4.70E-05	1.70E-06	6.13E-08
	2	6.15E-03	4.82E-04	3.78E-05	2.96E-06
	3	1.87E-02	2.57E-03	3.51E-04	4.81E-05
	4	4.48E-02	9.47E-03	2.00E-03	4.24E-04
	5	9.15E-02	2.77E-02	8.37E-03	2.53E-03

Conventionally, we are using 4 digits of Arabic numbers (symbol size $s = 10$) for a PIN code. Under a phishing free environment, the code is protected by its key space of size 10^4 . Without this virtual password protection, one successful phishing attack will completely invalid the PIN code. It is clear that our codebook approach can significantly decrease the chance of breaking the protection after a few successful phishing. It is safe to assume that after a few tries under the phishing attacks the victim will become specious and stop responding to the phisher. According to the table above, after 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 the account 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 to compromise 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 practices 64 is a reasonable symbol size. We have the following results.

TABLE 2: THE CHANCE OF BREAKING THE PASSWORDS UNDER PHISHING ATTACKS (SYMBOL SIZE =64)

symbol size s = 64		n: length of the password			
		4	6	8	10
p: number of successful phishing attacks	0	5.96E-08	1.46E-11	3.55E-15	8.67E-19
	1	9.84E-07	9.76E-10	9.68E-13	9.60E-16
	2	5.03E-06	1.13E-08	2.53E-11	5.68E-14
	3	1.60E-05	6.39E-08	2.56E-10	1.02E-12
	4	3.92E-05	2.45E-07	1.53E-09	9.59E-12
	5	8.14E-05	7.34E-07	6.62E-09	5.97E-11
	6	1.51E-04	1.85E-06	2.28E-08	2.80E-10
	7	2.58E-04	4.14E-06	6.64E-08	1.07E-09
	8	4.13E-04	8.40E-06	1.71E-07	3.47E-09
	9	6.30E-04	1.58E-05	3.97E-07	9.97E-09
	10	9.23E-04	2.81E-05	8.53E-07	2.59E-08

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.

VI. μ TESLA AUTHENTICATION

In previous sections, we discussed how to use the virtual password to defend against Phishing, key loggers, and shoulder-surfing. In this section, we propose another scheme to guard against phishing attacks by allowing the user to authenticate the server and adopting μ TESLA to provide freshness of the server key. The purpose of this scheme can be also used for authentication of the server before re-keying of the previous scheme. This μ TESLA scheme can be very useful when the web browser or other client side applications, such as the helper-application in our virtual password, can have some authentication function implemented. This scheme can also be used for protecting Phishing via emails.

μ 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, \dots, K_0 by randomly choosing the K_m and then producing the $K_{n-1} = F(K_n)$ where $n=0, 1, \dots, m$. 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 $E_{K_{n-1}}(K_n)$ 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_{n-1}) which it holds to get the K_n , and then using this K_n to verify if this sign-on screen is from the right server in the following way: if $F(K_n) = Key$ 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, \dots, 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 μ TESLA 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.

VII. IMPLEMENTATION AND SURVEY

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 shoulder-surfing attacks. 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. 2 and 3 demonstrate the usability testing results:

From Figs. 2 and 3, 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.

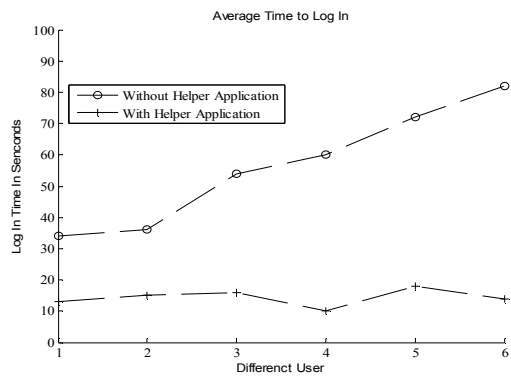


Fig. 2 Average log in time

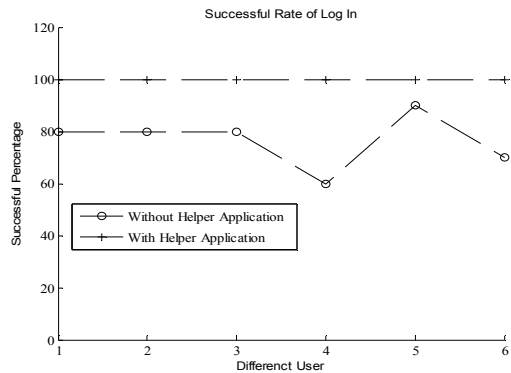


Fig. 3 Average Successful Rate (in percentage)

VIII. 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 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 a function serving as a system recommended function and provide security analysis. 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.

REFERENCE

- [1] T. Dierks and C. Allen. The TLS Protocol— Version 1.0. IETF RFC 2246, January 1999.
- [2] <http://en.wikipedia.org/wiki/Phishing>
- [3] Anti-phishing working group. <http://www.antiphishing.org>.
- [4] http://en.wikipedia.org/wiki/Key_logger
- [5] <http://www.eweek.com/article2/0,1895,1940623,00.asp>
- [6] B. Ross, C. Jackson, N. Miyake, D. Boneh, J. Mitchell, "Stronger Password Authentication Using Browser Extensions," Proceedings of 14th USENIX Security Symposium.
- [7] E. Gabber, P. Gibbons, Y. Matias, and A. Mayer, "How to make personalized web browsing simple, secure, and anonymous," Proceedings of Financial Crypto '97, volume 1318 of LNCS. Springer-Verlag, 1997.
- [8] E. Gabber, P. Gibbons, D. Kristol, Y. Matias, and A. Mayer, "On secure and pseudonymous user-relationships with multiple servers," ACM Transactions on Information and System Security, 2(4):390–415, 1999.
- [9] E. Jung. Passwordmaker. <http://passwordmaker.mozdev.org>.
- [10] J. la Poutr' e, "Password composer," <http://www.xs4all.nl/?jlpoutre/BoT/Javascript/PasswordComposer/>.
- [11] J. R. Levine, "A Flexible Method to Validate SMTP Senders in DNS," Apr. 2004. http://www1.ietf.org/proceedings_new/04nov/IDs/draft-levine-fsv-01.txt.
- [12] V. A. Brennen, "Cryptography Dictionary," vol. 2005, 1.0.0 ed, 2004.
- [13] <http://www.bell-labs.com/project/lpwa/>
- [14] E. Damiani et al., "Spam Attacks: P2P to the Rescue," Proceedings of Thirteenth International World Wide Web Conference, pages 358–359, 2004.
- [15] M. Abadi, L. Bharat, and A. Marais, "System and method for generating unique passwords," US Patent 6,141,760, 1997.
- [16] M. Kuhn, "Probability theory for pickpockets – ec-PIN guessing," Available at <http://www.cl.cam.ac.uk/?mgk25/>, 1997.
- [17] C. Herley and D. Florencio, "How To Login From an Internet Caf' e Without Worrying About Keyloggers," Proceedings of Symposium on Usable Privacy and Security (SOUPS) '06
- [18] <http://www.citibank.co.jp/en/service/cap/virtualpad/>
- [19] http://obr.typepad.com/financial_innovations/2005/11/ing_direct_adds.html
- [20] M'OLLER, B. Schw'achen des ec-PIN-Verfahrens. Available at <http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/moeller>, Feb. 1997. Manuscript.
- [21] A. Herzberg and A. Gbara, "Trustbar: Protecting (even naive) web users from spoofing and phishing attacks," Cryptology ePrint Archive, Report 2004/155, 2004. <http://eprint.iacr.org/2004/155>.
- [22] S. Wiedenbeck, J. Waters, L. Sobrado, and J. Birget, "Design and evaluation of a shoulder-surfing resistant graphical password scheme," Proc. of the working conference on Advanced visual interfaces, Venezia, Italy.
- [23] V. Roth, K. Richter, and R. Freidinger, "A PIN-entry method resilient against shoulder-surfing," Proc. of the 11th ACM Conference on Computer and Communications Security, 2004, 236-245.
- [24] IETF. MTA Authorization Records in DNS (MARID), June 2004. <http://www.ietf.org/html.charters/OLD/marid-charter.html>.
- [25] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage," Proceedings of the 12th Annual Network and Distributed System Security Symposium, 2005.
- [26] G. T. Wilfong, "Method and apparatus for secure PIN entry," US Patent #5,940,511, United States Patent and Trademark Office, May 1997. Assignee: Lucent Technologies, Inc. (Murray Hill, NJ).
- [27] J. Mason, "Filtering Spam with SpamAssassin," Proceedings of HEANet Annual Conference, 2002.
- [28] D. Stinson, Cryptography Theory and Practice, Second Edition.
- [29] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A Bayesian Approach to Filtering Junk E-Mail. In Learning for Text Categorization," The 1998 Workshop, May 1998
- [30] T.A. Meyer and B. Whateley, "SpamBayes: Effective open-source, Bayesian based, email classification system,"
- [31] MAPS. RBL - Realtime Blackhole List, 1996. http://www.mail-abuse.com/services/mds_rbl.html.
- [32] The Spamhaus Project. The Spamhaus Block List. <http://www.spamhaus.org/sbl/>.
- [33] Netcraft. Anti-Phishing Toolbar. http://news.netcraft.com/archives/2004/12/28/netcraft_antiphishing_tool%bar_available_for_download.html.
- [34] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: security protocols for sensor networks. Wireless Networking, 8(5):521–534, 2002